WSMO-Lite and hRESTS: lightweight semantic annotations for Web services and RESTful APIs

Journal Item

WSMO-Lite and hRESTS: lightweight semantic annotations for Web services and RESTful APIs

Journal Article

ARTICLE IN PRESS

Web Semantics: Science, Services and Agents on the World Wide Web ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs

Dumitru Roman [a,*], Jacek Kopecký [b], Tomas Vitvar [c], John Domingue [e], Dieter Fensel [d]

[a] SINTEF, Forskningsveien 1, 0314 Oslo, Norway

[b] School of Computing, University of Portsmouth, Buckingham Building, Lion Terrace, Portsmouth, PO1 3HE, UK

[c] Institut für Informatik, University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria

[d] STI Innsbruck, University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria

[e] Knowledge Media Institute, The Open University, Walton Hall, MK7 6AA Milton Keynes, UK

ARTICLE INFO

Article history:
Received 17 February 2014
Received in revised form
24 November 2014
Accepted 26 November 2014
Available online xxxx

Keywords:
WSMO-Lite
SAWSDL
Web services
RESTful services

ABSTRACT

Service-oriented computing has brought special attention to service description, especially in connection with semantic technologies. The expected proliferation of publicly accessible services can benefit greatly from tool support and automation, both of which are the focus of Semantic Web Service (SWS) frameworks that especially address service discovery, composition and execution. As the first SWS standard, in 2007 the World Wide Web Consortium produced a lightweight bottom-up specification called SAWSDL for adding semantic annotations to WSDL service descriptions. Building on SAWSDL, this article presents WSMO-Lite, a lightweight ontology of Web service semantics that distinguishes four semantic aspects of services: function, behavior, information model, and nonfunctional properties, which together form a basis for semantic automation. With the WSMO-Lite ontology, SAWSDL descriptions enable semantic automation beyond simple input/output matchmaking that is supported by SAWSDL itself. Further, to broaden the reach of WSMO-Lite and SAWSDL tools to the increasingly common RESTful services, the article adds hRESTS and MicroWSMO, two HTML microformats that mirror WSDL and SAWSDL in the documentation of RESTful services, enabling combining RESTful services with WSDL-based ones in a single semantic framework. To demonstrate the feasibility and versatility of this approach, the article presents common algorithms for Web service discovery and composition adapted to WSMO-Lite.

1. Introduction

The emergence of service-oriented computing has brought special attention to the area of service modeling. Service descriptions are a fundamental element that enables such tasks as service discovery, composition and mediation. As argued by [1], semantic interoperability is crucial for Web services, and the technologies of the Semantic Web have the potential to provide the needed level of interoperability. In the recent years, research into semantic Web service (SWS) models and descriptions has received much attention and funding: the major research projects include the US-based OWL-S initiative,[1] and the European projects DIP, SUPER and SOA4All.[2] In these and other projects, researchers have proposed several frameworks for semantic Web services (SWS), especially OWL-S, WSMO, and WSDL-S.

OWL-S [2] and WSMO [3] embody a *semantics-first* approach to semantic modeling of Web services: in both frameworks, the semantic description of a service is conceptually independent of the underlying technical description (e.g. WSDL [4]), and there is a *grounding* mechanism to connect the semantics with the technical descriptions. In contrast, WSDL-S [5] attaches semantic annotations directly to the underlying technical descriptions, in effect building on the Web service model of WSDL.

Service-oriented computing, revolving around the so-called WS–∗ family of technologies (based on the SOAP protocol [6] and the WSDL service description language), is heavily driven by standardization. The approach embodied by WSDL-S turned out to

* Corresponding author.
E-mail addresses: dumitru.roman@sintef.no (D. Roman), jacek.kopecky@port.ac.uk (J. Kopecký), tomas@vitvar.com (T. Vitvar), john.domingue@open.ac.uk (J. Domingue), dieter.fensel@sti2.at (D. Fensel).

[1] http://www.daml.org/services/owl-s/.

[2] dip.semanticweb.org, ip-super.org, soa4all.eu.

http://dx.doi.org/10.1016/j.websem.2014.11.006
1570-8268/

be an acceptable common ground for both the research community and industry; based on WSDL-S, the World Wide Web Consortium (W3C) produced in 2007 its Recommendation called SAWSDL—Semantic Annotations for WSDL and XML Schema [7]. SAWSDL is not a fully-fledged SWS framework; instead it only provides hooks in WSDL where semantic annotations can be attached, leaving the specification and standardization of concrete service semantics for later. Modeling service semantics, within the framework of SAWSDL, is the focus of our work.

In addition to WS–∗, in recent years another approach to service-oriented computing has started gaining traction: *RESTful Web services* [8], often also called Web APIs. RESTful services are a common part of Web applications, as the applications' functionalities are increasingly used by programmatic clients, such as other Web applications. RESTful services are especially present in Web applications that use rich *Ajax*[3] user interfaces, where the JavaScript code running in the browser is a programmatic client, for which the Web application must provide an appropriate access interface. RESTful Web services use native Web technologies around HTTP [9], the Web architecture [10], and standard data formats, resulting in effective integration with the Web. RESTful services are proliferating in part because the Web technologies used by RESTful Web service are perceived as simpler than the WS–∗ technologies [11].

In [11] the authors analyze the architectural differences between WS–∗ and RESTful services. Among the key differences between WS–∗ and RESTful services is their client-facing structure: where a WS–∗ service exposes a single endpoint that handles all the operations of the service, a non-trivial RESTful service exposes a number of independently meaningful and addressable resources.

Irrespective of the types of services (WS–∗ vs. RESTful), the availability of service annotations is a prerequisite for service automation in discovery, composition, mediation, etc. Treating service annotations uniformly for both WS–∗ and RESTful services would not only enable better interoperability between them but would also enable technologies for service automation to be applicable across both types of services. In our work, we reconcile the two different kinds of services and we show that most semantic automation algorithms can disregard the difference, eventually enabling more scalable and robust mechanisms for WS–∗ and RESTful services tasks.

To achieve this reconciliation, we define hRESTS and MicroWSMO – two HTML microformats that mirror WSDL and SAWSDL over the HTML documentation of RESTful services – enabling us to create WSMO-Lite semantic descriptions of RESTful services. Furthermore, we develop a concrete ontology of service semantics (WSMO-Lite) for use with the standard SAWSDL, and we broaden the applicability of SAWSDL to RESTful services and Web APIs (through hRESTS and MicroWSMO). Finally, to demonstrate the viability of our SWS approach, we adapt automation algorithms for service discovery and composition to WSMO-Lite.

The overall purpose of this paper is to give a comprehensive overview of the WSMO-Lite approach for reconciling WS–∗ and RESTful services, to demonstrate the feasibility of the approach, and to serve as an entry and reference point for interested parties in using the WSMO-Lite approach.

Parts of this paper are based upon, and extend work reported in [12,13]. [12] introduced the annotation of WSDL services with WSMO-Lite, and [13] introduced the hRESTS microformat, however, no previous publication provided this comprehensive overview of the WSMO-Lite approach, or the feasibility analysis on how SWS tasks can be applied to the approach. To the extent of our knowledge, WSMO-Lite is the first principled effort focused on defining an ontology of service semantics for SAWSDL service descriptions.

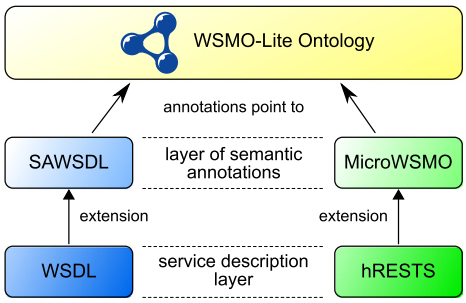


Fig. 1. Semantic Web service descriptions with WSMO-Lite.

The rest of this article is organized as follows. Section 2 provides an overview of the WSMO-Lite framework for lightweight semantic service descriptions, and highlights some design considerations important for our work. Section 3 introduces a minimal service model that captures the common structure of Web services and bridges the worlds of WS–∗ and RESTful Web services. Section 4 formally defines the WSMO-Lite ontology for service semantics, showing how it fits WSDL and SAWSDL. Section 5 shows how we apply SAWSDL and WSMO-Lite to RESTful services—it defines the two microformats, hRESTS and MicroWSMO, for annotating RESTful services' HTML documentation. Section 6 presents concrete algorithms for service discovery and composition, adapted to WSMO-Lite from earlier SWS research. Section 7 presents an analysis of feasibility of the overall approach from different angles (tooling support, viability), Section 8 discusses related work and the limitations of WSMO-Lite, and Section 9 concludes the article with a summary of the core contributions and with a discussion of future work.

2. Overall approach and design principles

Fig. 1 shows the core technologies that make up our framework. Our work stems from the standardization of SAWSDL; as illustrated on the left-hand side of the figure, SAWSDL extends the service description language WSDL with semantic annotations. For the content of the annotations, we define the *WSMO-Lite ontology* of service semantics, where we identify the kinds of service semantics that are necessary to support SWS automation.

The right-hand side of the figure shows our technologies for describing and semantically annotating RESTful services: *hRESTS and MicroWSMO*. WSMO-Lite builds on a unified minimal service model that extracts the concepts common in WS–∗ and RESTful Web services. Based on this model, hRESTS (HTML for RESTful Services) is a microformat[4] for structuring common HTML documentation of RESTful APIs to make it machine-processible, analogously to how WSDL provides machine-processible descriptions on the WS–∗ side.[5] MicroWSMO (Microformat for WSMO-Lite) is an extension of hRESTS that adds SAWSDL annotation properties, where WSMO-Lite semantics can be attached.

Our work is guided by the following design principles:

Proximity to underlying standards. In semantics-first frameworks (OWL-S, WSMO), the semantic description of a service is

[3] http://en.wikipedia.org/wiki/Ajax_(programming).

[4] Microformats are an approach for annotating mainly human-oriented Web pages so that key information is machine-readable [14].

[5] There is currently no accepted equivalent of WSDL for RESTful services; the Web Application Description Language (WADL, [15]) is a RESTful (resource-oriented) alternative to WSDL, but it is not commonly accepted.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

conceptually independent of the underlying technical description (e.g. WSDL), and there is a "grounding" mechanism to connect them. In contrast, WSMO-Lite follows the spirit of WSDL-S and SAWSDL and breaks up the semantics of Web services into pieces that attach directly to SAWSDL. With this approach, the semantic description does not get separated from the underlying WSDL, as would be the case with OWL-S or WSMO.

Inclusion of RESTful services. In order for semantic automation algorithms to be able to work uniformly with WS–∗ as well as RESTful services, WSMO-Lite defines a minimal RDF service model that captures the common structure of Web services. WSDL descriptions annotated with SAWSDL, or hRESTS description annotated with MicroWSMO, are parsed as RDF instances of the service model. Working on top of this RDF view of the underlying descriptions, most automation algorithms need not distinguish between the two kinds Web services; only the algorithms that involve actual service invocations must use the appropriate communication technologies which differ between WS–∗ and RESTful services.

Lightweight minimality. In the spirit of SAWSDL, WSMO-Lite aims to be minimalistic and lightweight: (a) it defines a very small vocabulary for service semantics, and a minimal service model common to WS–∗ and RESTful services, (b) these are defined in the most basic Semantic Web ontology language, RDFS [16], which has very limited reasoning requirements but can easily accommodate more expressive languages, especially including languages for logical expressions and rules; (c) WSMO-Lite builds on WSDL, which is already well-known to Web services practitioners; (d) the choice of microformats as the mechanism for describing RESTful services limits the need for new syntax, and (e) the two microformats are also tightly scoped to fit already existing service documentation. In effect, WSMO-Lite adds very few new constructs on top of the underlying technologies that are already well-known, and it carries no inherent requirements on reasoning power.

Modularity. WSMO-Lite distinguishes between four types of service semantics: the service's function, behavior, information model, and its nonfunctional properties. Together, these types of service semantics support the automation of all the major service consumer's tasks: discovery, composition, negotiation, ranking, invocation, etc.[6] WSMO-Lite annotations are modular: in any given service-oriented system, increasing needs for automation can be met incrementally by adding and refining various types of semantic annotations. For example, when the number of services becomes hard to manage, functional annotations can be added to support service discovery and composition; when many services are being composed together, information model annotations can be added to facilitate data mediation; and later nonfunctional properties can be added so that the system can adaptively respond to service failures with replacements that have similar nonfunctional parameters (e.g. quality of service, policy, price) as the service that failed.

3. Service model analysis

Our goal is to support both WS–∗ and RESTful Web services. In order for semantic automation algorithms to be able to work uniformly with the two kinds of services, we build WSMO-Lite on a minimal RDFS model that captures the common structure of Web services.

The minimal service model is derived from relevant work on service modeling and description, mainly from the standard for Service-Oriented Architecture Reference Model (SOA RM) and from WSDL. Since these specifications are part of the WS–∗ family of standards, we also show that our service model is nevertheless appropriate for RESTful services as well. The model and the demonstration of its applicability to RESTful services are the main contributions of this section.

For automated processing, the minimal service model serves as a layer of abstraction over the underlying concrete Web service description languages: WSDL on the WS–∗ side, and HTML documentation for RESTful services. These service descriptions are annotated with semantics, as discussed in the following Sections 4 and 5. Service descriptions can be parsed into RDF in terms of the service model, and then semantic automation algorithms (for discovery, composition, etc.) can process the semantic annotations without distinguishing WS–∗ services from RESTful services.

Below, in Section 3.1, we analyze relevant service modeling and description specifications that underlie our WSMO-Lite minimal service model, which is defined in Section 3.2. Finally in Section 3.3, we show that this model also naturally applies to RESTful services.

3.1. Existing service models

The term "service" in its modern usage comes from economics. The services sector now dwarfs agriculture and manufacturing, the original main sectors of the economy; its boom has recently spawned the field of Services Science [17], which deals with terms such as *exchange of goods* and *economic entities*. In IT, there have been several efforts to define service models and conceptualize services and service-oriented architectures, activities initiated in academic research (e.g. [18–20]) as well as in the context of standardization bodies (e.g. [21]).

The term "Web service" started in association with a set of concrete technologies for distributed computing, especially WSDL and SOAP, as shown in [22]. Service-oriented architecture (SOA) is an abstraction of Web services: it is oriented toward large-scale distributed systems, and it sheds the technology bias of "Web services". SOA is currently best articulated by the OASIS consortium's industry-standard Reference Model for Service Oriented Architecture (SOA RM, [21]), which defines a service as follows:

> *A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.*

The SOA RM investigates a number of aspects of services; it identifies the top-level six aspects to be *Service description*, *Visibility*, *Execution context*, *Real-world effect*, *Contract and policy*, and *Interaction*. A service in SOA is generally not a single artifact, instead it is a concept useful for manageability. From the point of view of a client, the constituent elements of a service are its network location (part of the *Visibility* aspect) and the service description.

Service description[7] is key to automation, because a client, broker, or other intermediary only has service descriptions to

[6] The service consumer's tasks are identified in the Semantic Execution Environment Tech. Committee (SEE TC) at OASIS, the primary standardization body for WS–∗; see http://oasis-open.org/committees/tc_home.php?wg_abbrev=semantic-ex.

[7] In this section we *emphasize with italics* the terms defined by the SOA RM.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

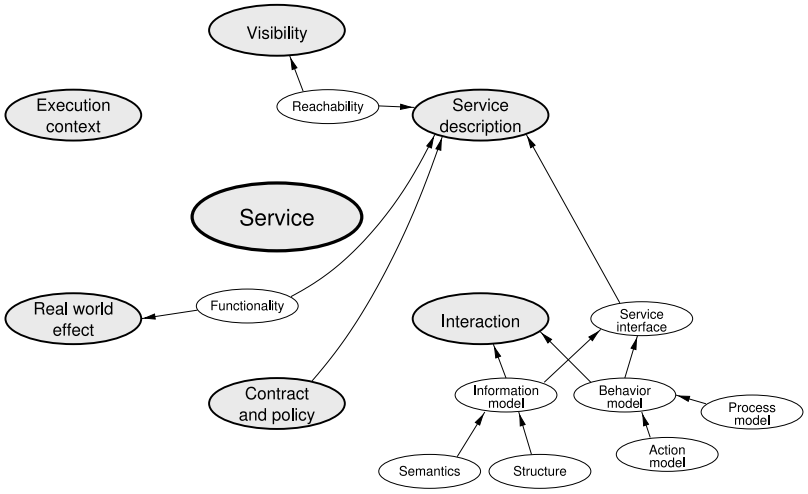


Fig. 2. Subset of OASIS SOA RM relevant for the WSMO-Lite service model.

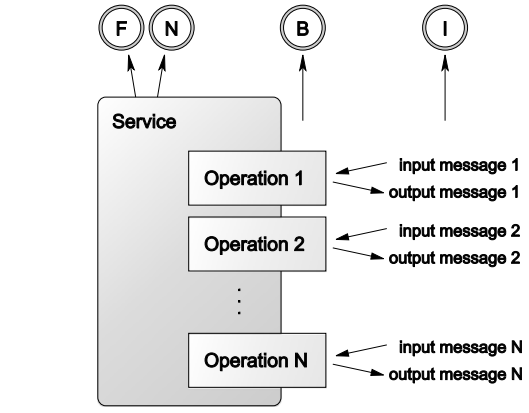
guide it in selecting and using the available Web services. Fig. 2 (a simplified combination of Figs. 6 and 9 from SOA RM, highlighting in gray the six top-level aspects of services) details the parts of the SOA RM that must be captured in a machine-readable description: a service has a certain *functionality*, which implements the service capabilities that achieve the *real-world effect*; it may have additional constraints (*contracts and policies*); it is *reachable* (commonly through a computer network, at a given location that gives the service *visibility*); and it has a *service interface*, made up of an *information model* and a *behavior model*, both involved in *interactions* between the service and its clients.

The *information model* of a service interface characterizes the information that may be exchanged with the service. It specifies the *semantics* (i.e., the meaning) of the data, and its *structure* and form. The *behavior model* describes the *actions* (operations) that may be invoked on the service, and the *process* that defines the possible order(s) in which the actions make sense. In the words of the SOA RM, "the process model characterizes the temporal relationships and temporal properties of actions and events associated with interacting with the service". The interaction side of the service is the most detailed part of service description in SOA RM because it represents the majority of the service's interface for use by clients.

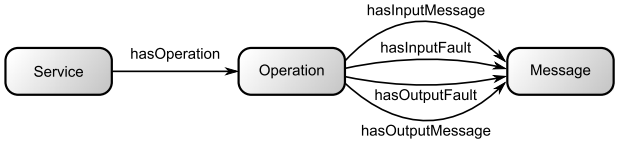
Information about operations, message structures, communication protocols and message exchange patterns, and physical service access points (service *reachability*), is already part of technical descriptions such as WSDL; in this article we do not study the semantics of these underlying technical descriptions. In our case, for automation using semantics, we want to represent the semantics of the remaining aspects not covered in the underlying technical descriptions.

Inspired by the distinctions made by [23], we group the remaining aspects of SOA RM service description into four orthogonal parts:

- Functional description specifies service *functionality*, that is, what a service can offer to its clients when it is invoked.
- Nonfunctional description defines any *contract and policy* parameters of a service, or, in other words, incidental details specific to the implementation or running environment of the service.
- Behavioral model specifies the *process* (in other words, the ordering of operations) that a client needs to follow when consuming a service's functionality.
- *Information model* defines the input, output and fault messages of the actions.



(a) Structural view with functional, nonfunctional, behavioral and information semantics.



(b) Conceptual view.

Fig. 3. WSMO-Lite Web service description model.

Functional and nonfunctional semantics are directly properties of a service. Behavioral semantics tie to service operations. Finally, information semantics tie to the data that a service communicates with—to the input, output and fault messages of the operations. This leads us to a service model extracted from what already exists in the underlying technical descriptions: a service has a number of operations, each of which may have input, output and fault messages. The following subsection defines the concrete terms of the model, along with an RDFS ontology that captures them.

3.2. WSMO-Lite service model

The WSMO-Lite service description model is a straightforward simplification of the structure of WSDL, following the design principles of *proximity to underlying standards* and *lightweight minimality*. Fig. 3 shows two views of the model. The upper part (a) of the figure shows the structure of a service description, separating the non-semantic structure below from the semantic

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

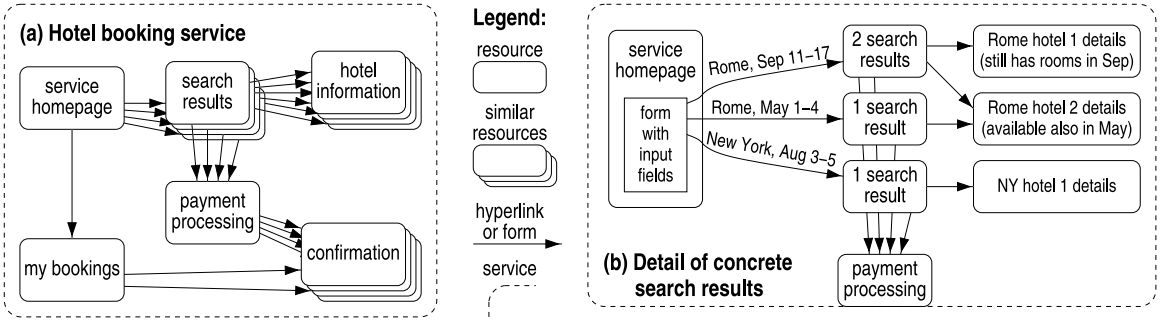


Fig. 4. Structure of an example RESTful hotel reservation service: (a) showing types of resources, (b) showing details of example search results resources that make up a part of the service.

Table 1
Mapping of RESTful services to WSMO-Lite service model.

RESTful services	WSMO-Lite service model
Service (*group of resources*)	Service
Resource	*(disregarded on the semantic level)*
Resource method	Operation
Method request/response	Operation input/output
Hyperlink	*(treated as part of message data)*

aspects on top. For expressing the semantic aspects, we define in Section 4 the WSMO-Lite service semantics ontology. The lower part (b) of the figure shows the concepts that make up the service model, along with the relationships between them.

The WSMO-Lite service model is rooted in the concept of *service*. While some technologies, such as WSDL, abstract the interface of a service from the service itself, so that the interface definition is reusable between multiple services, in our service model the interface would only constitute an indirection between a service and its semantic annotations, with no added value. Therefore, our service model does not separate the interface from the service.

A Web service is a collection of operations (at least one). Operations have input and output messages (with inputs and outputs viewed from the side of the service), and potentially input and output faults as well. Which of these messages should be present depends on the operation's *message exchange pattern*: the most common one is *request–response*, where an operation has a single input (request) message followed at run-time either by a single output (response) message or by an output fault. Some advanced message exchange patterns can also include input faults (faults going from the client to the service), as discussed especially in [24].

The service model is mirrored straightforwardly in an RDFS [16] ontology. It contains three classes: wl:Service, wl:Operation and wl:Message,[8] and five properties: wl:hasOperation, wl:hasInputMessage, wl:hasOutputMessage, wl:hasInputFault and wl:hasOutputFault.[9] Instances of this RDFS ontology are created as the result of parsing WSDL or hRESTS service descriptions, effectively acting as an RDF view of those descriptions.

3.3. Modeling RESTful services

RESTful Web services are hypermedia applications consisting of interlinked resources (like Web pages) that are oriented toward machine consumption. In their structure and behavior, RESTful Web services can be very much like common Web sites [8]. The hypertext nature of RESTful Web services seems to differ from the service model discussed so far: it requires that services be decomposed into Web resources which are preferably units of data, whereas the WS–∗ model decomposes services into operations, i.e. units of function. In this section, we show how these approaches can be reconciled, according to our design principle of *inclusion of RESTful services*.

From the Architecture of the Web [10] and from the REST architectural style [26], we can extract the following concepts inherent in RESTful services: a *resource*, identified by a URI that also serves as the endpoint address where clients can send requests; every resource has a number of *methods* (in HTTP [9], the most-used methods are GET, HEAD, POST, PUT and DELETE) that are invoked by means of request/response message exchanges. The messages can carry *hyperlinks*, which point to other resources and which the client can navigate when using the service. A hyperlink can simply be a URI, or it can be a *form* which specifies not only the URI of the target resource, but also the method to be invoked and the structure of the input data.

Note that the architecture of the Web contains no formal concept of a *service* as such; a service is a grouping of resources that is useful for developing, advertising and managing related resources.

While the resources of the service (the *nouns*) form a hypermedia graph, the interaction of a client with a RESTful service is a series of operations (the *verbs* or *actions*) where the client sends a request to a resource and receives a response that may link to further useful resources. The hypermedia graph (the links between resources) guides the sequence of operation invocations, but the meaning of a resource is independent of where it is linked from; the same link or form, wherever it is placed, will always lead to the same action. Therefore, even though this may be counterintuitive, the operations of a RESTful Web service can be considered independently from the graph structure of the hypertext.

In Table 1, we summarize the mapping from the Web-architecture-based model of RESTful services into the WSMO-Lite minimal service model. Effectively, an automated client (such as a semantic automation system) can view RESTful services through the model from Fig. 3, with the operations being the methods available on the resources that constitute the RESTful services. This common view allows us to support WS–∗ and RESTful services in WSMO-Lite without regard to their technological and architectural differences.

Fig. 4(a) depicts an example RESTful hotel booking service, with its resources and the links among them; we use this synthetic example to demonstrate how a hypermedia service can naturally be viewed as a set of operations.

The "service homepage" is a resource with a stable address and information about the other resources that make up the service. It

[8] The prefix wl denotes the WSMO-Lite namespace http://www.wsmo.org/ns/wsmo-lite# (which is also the location of the ontology).

[9] Note that WSDL, from which we extract the WSMO-Lite service model, also has an RDF mapping (and a simple OWL ontology) defined in [25]; its structure reflects the complexity of WSDL and therefore it is unsuitable for a lightweight framework such as WSMO-Lite, hence the need for our own ontology in the WSMO-Lite namespace.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

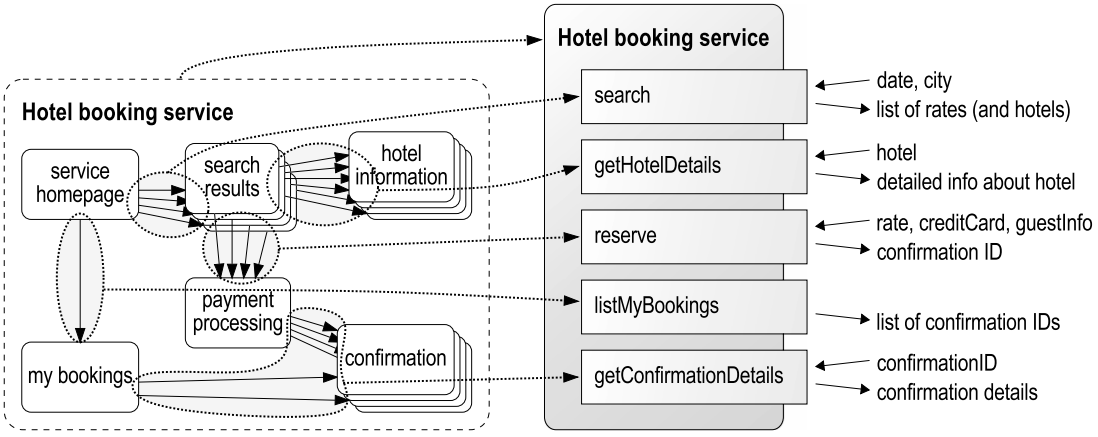


Fig. 5. Extracting operations of the example service from Fig. 4.

serves as the initial entry point for client interaction. In a human-oriented Web application, this would be the homepage, such as http://hotels.example.com/.

The homepage resource of our example service contains a form for searching for available hotels, given a number of guests, a start date, the duration of the stay, and a location. The search form serves as a parametrized hyperlink to search results resources, as shown in Fig. 4(b), one resource per every unique combination of the input data—the form prescribes how to create a URI that contains the input data; the URI then identifies a resource with the search results.

In our example, search results are presented as a list of concrete rates available at the hotels in the given location, for the given dates and the number of guests, as also shown in Fig. 4(b). Each item of the list contains a link to further information about the hotel (e.g. the precise location, star rating and other data), and a form for booking the rate, which takes as input the payment details (such as credit card information) and an identification of the guest(s) who will stay in the room. The form data is submitted (with the POST method) as a booking request to a payment resource, which processes the booking and returns a newly created confirmation resource. The content of the confirmation can serve as a receipt.

The service homepage resource also links to "my bookings", a resource listing the bookings of the current user (who is identified through a suitable authentication functionality). This resource links to the confirmations of the bookings made by the user. With such a resource available to them, client applications need not store the information about performed bookings locally.

Together, all the resources we have described here form the hotel booking service. So far, our description of the example hotel reservation service has been based on the hypermedia aspect: we described the resources and how they link to each other. In Fig. 5, we extract the operations present in the example service into the model from Fig. 3, demonstrating the mapping from Table 1. The search form in the homepage represents a search operation, the hotel information pages linked from the search results can be viewed as an operation for retrieving hotel details, the reservation form for any particular available rate becomes a reservation operation, and so on.

We have shown how we can view RESTful services as sets of operations, in concord with the comparison of WS–∗ and RESTful services in [11] which puts RESTful services in the *Remote Procedure Calls* integration style. Because we can see RESTful services as sets of operations, we can apply the same kinds of semantic descriptions as we apply to WS–∗ services; in the following section, we discuss how we represent service semantics on top of this model.

4. WSMO-Lite ontology of service semantics

In the analysis of the service model in Section 3.1, we distinguished between four types of semantics that can be used in order to support automation: functional, nonfunctional, behavioral and information model semantics. This section focuses on formalizing these semantics and how they apply in SAWSDL and in our minimal service model.

In particular, we start with a formalization of the service semantics in Section 4.1. Section 4.2 formalizes the SAWSDL annotation mechanism for attaching semantics to service descriptions in the minimal service model and Section 4.3 shows how the annotations should apply to the more complex model of WSDL.

4.1. Service semantics representation in WSMO-Lite

Informally, the four types of service semantics are represented in the WSMO-Lite service ontology as follows:

- Information semantics are represented using domain *ontologies*, which are also involved in the descriptions of the other types of semantics.
- Functional semantics are represented in WSMO-Lite with *capabilities* and/or functionality *classifications*. A capability defines *preconditions* which must hold in a state before the client can invoke the service, and *effects* which hold in a state after the service invocation. Functionality classifications define the service functionality using some classification ontology (i.e., a hierarchy of *categories*).[10]
- Nonfunctional semantics are represented using an ontology that semantically captures some policy or other nonfunctional properties.
- Behavioral semantics are represented by annotating the service operations with functional descriptions, i.e., capabilities and/or functionality classifications. In later sections, we demonstrate how the functional annotations of operations serve for the ordering of operation invocations.

We formalize these terms below. Mainly, we define ontology, which is the fundamental building block for all types of semantic descriptions. We use a generic definition of ontology that encompasses the common definitions in literature; our definition is only as specific as necessary to capture core ontology elements needed for the purpose of this work, and it remains general enough for us

[10] The distinction of capabilities and categories is the same that is made by [1] between "explicit capability representation" (using taxonomies) and "implicit capability representation" through preconditions and effects.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

to be able to plug in various knowledge representation languages, such as RDFS [16], OWL [27] and RIF [28], as appropriate in any particular system. Further, we formalize classification and capability, which provide a functional description of services and operations.

The formalization allows us to be explicit about the terms we define and about how the terms apply to SAWSDL annotations (esp. in Table 2), and it is also useful for defining algorithms that process WSMO-Lite descriptions, such as the ones shown in Section 6.

Definition 4.1 (*Ontology*)**.** An ontology Ω is a four-tuple

$$\Omega = (C, R, E, I)$$

where C, R, E, I are sets that, in turn, denote classes (unary predicates), relations (binary and higher-arity predicates), explicit instances of classes and relations (extensional definition), and axioms (intensional definition) that describe how instances are inferred.

A particular axiom common in ontologies is the *subclass* relationship between two given classes c_1 and c_2: if c_1 is subclass of c_2 (written as $c_1 \subseteq c_2$), every instance of c_1 is also an instance of c_2. In general, the subclass relationship forms a partial order on the set of classes (it is a transitive, reflexive and antisymmetric binary relation). To indicate that an ontology contains a subclass axiom between the given classes c_1 and c_2, we write $(c_1 \subseteq c_2) \in I$. Along with the subclass relationship, ontologies may also contain a *subrelation* relationship ($r_1 \subseteq r_2$), defined analogously.

Definition 4.2 (*Classification*)**.** A classification is an ontology $\Omega_C(c_0) = (C, R, E, I)$ whose classes (members of C) represent categories of things. Classification categories form a subclass (subcategory) hierarchy with a single root c_0, i.e., every class in the ontology is either directly a subclass of c_0 (as captured by the subclass axioms within I), or it is a subclass by transitivity through a finite sequence of other classes.[11]

For the purposes of describing the different kinds of service semantics, we distinguish several sub-types of ontologies: an information model ontology (an ontology used as an information model in a service description) is denoted as $\Omega^I \equiv \Omega$; a functionality classification ontology with root $c_0 \in C$ (whose classes form a taxonomy of service functionalities), denoted as $\Omega^F(c_0) \equiv \Omega_C(c_0)$; and an ontology for nonfunctional semantics as $\Omega^N \equiv \Omega$, whose instances (members of E) are concrete nonfunctional descriptions.

Definition 4.3 (*Capability*)**.** A capability is a three-tuple

$$K = (\Sigma, \phi^{pre}, \phi^{eff})$$
$$\Sigma \subseteq V \cup C \cup R \cup E$$

where K (kappa) stands for the capability, ϕ^{pre} is a *precondition* which must hold in a state before the service (or operation) can be invoked, and ϕ^{eff} is the *effect*, an expression which is expected to hold in a state after the successful invocation. Preconditions and effects are defined as logical statements over members of Σ, a set of identifiers of elements from C, R, E of some ontology Ω^I complemented with a set V of variable names.

The formal concepts used to describe service semantics introduced above can be materialized as an ontology in RDFS. The WSMO-Lite service semantics ontology rather straightforward and consists of four classes: wl:FunctionalClassificationRoot, wl:NonfunctionalParameter, wl:Condition and wl:Effect; the namespace also includes the terms of the minimal service model. For further details of the syntax, the reader is referred to [30].

[11] Note that it may also be practical in some systems to use less-formal SKOS [29] concept schemes with hierarchies of *broader* and *narrower* concepts: the SKOS *narrowerTransitive* property would replace the subclass axiom $\subseteq$, and a SKOS *top concept* would serve the function of the classification root.

Table 2
WSMO-Lite service model annotations with SAWSDL properties.

Svc. model	Annotation type/value		Context	Type
Service	*mref*	ϕ^{pre} or ϕ^{eff}	$K = (\Sigma, \phi^{pre}, \phi^{eff})$	F
Service	*mref*	$x \in C$	$\Omega^F(c_0) = (C, R, E, I)$	F
Service	*mref*	$x \in C \cup R \cup E$	$\Omega^N = (C, R, E, I)$	N
Operation	*mref*	ϕ^{pre} or ϕ^{eff}	$K = (\Sigma, \phi^{pre}, \phi^{eff})$	B
Operation	*mref*	$x \in C$	$\Omega^F(c_0) = (C, R, E, I)$	B
Message	*mref*	$x \in C \cup R$	$\Omega^I = (C, R, E, I)$	I
Message	*lift*	$f(data) \rightarrow X \subseteq E$	$\Omega^I = (C, R, E, I)$[a]	I
Message	*lower*	$g(X \subseteq E) \rightarrow data$	$\Omega^I = (C, R, E, I)$[a]	I

[a] As explained in the text of Section 4.2, the symbols f, g and *data* are not formalized any further.

4.2. Attaching semantics to the service model using SAWSDL RDF properties

The semantic concepts defined in the previous subsections are used to express the semantics of concrete services. The resulting semantic descriptions are used to annotate the underlying non-semantic descriptions (such as WSDL) using the standard SAWSDL properties *model reference*, *lifting schema mapping* and *lowering schema mapping*. After parsing the underlying non-semantic descriptions into an RDF form that follows the minimal service model from Section 3, the semantic annotations translate into the respective SAWSDL RDF properties, sawsdl:modelReference, sawsdl:liftingSchemaMapping and sawsdl:loweringSchemaMapping. Below, we first informally describe the SAWSDL properties, and then we define a simple formalization of the annotations, in relation to the minimal service model.

In SAWSDL, the main annotation property is a *model reference*, which points from any WSDL component to the associated semantics. In the context of WSMO-Lite, a model reference on a service can point to a description of the service's functional and nonfunctional semantics; a model reference on an operation points to the operation's part of the behavioral semantics description; and a model reference on a message points to the message's counterpart(s) in the service's information model ontology.

Each concrete model reference value is always identified with a URI. Multiple values of a model reference on a single component all apply to the component; for example, a service can have some nonfunctional properties, pointers to functionality categories, and preconditions and effects which together make up the capability of the service.

The other two SAWSDL properties, *lifting schema mapping* and *lowering schema mapping*, are necessary to connect semantic clients with the message-structure-oriented Web services. A semantic client works on the semantic level, with RDF data. In contrast, Web services and their clients usually exchange messages in XML or in a similar non-semantic structured data format. In order to enable the client to communicate with actual Web services, its semantic data must be *lowered*[12] into the expected input messages, and the data coming from the service in its output messages must be *lifted* back up to the semantic level. The lifting and lowering schema mapping properties are used to associate messages with appropriate transformations between the underlying technical format such as XML and a semantic knowledge representation format such as RDF. Both properties take as values the URIs of documents that define the lifting or lowering transformations.

Table 2 formalizes the content of the annotations on our service model. The first column specifies the service model component that is being annotated, the second column specifies the annotation property (model reference, lifting or lowering schema mapping),

[12] SAWSDL layers semantics on top of syntax, hence *lifting* and *lowering*.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

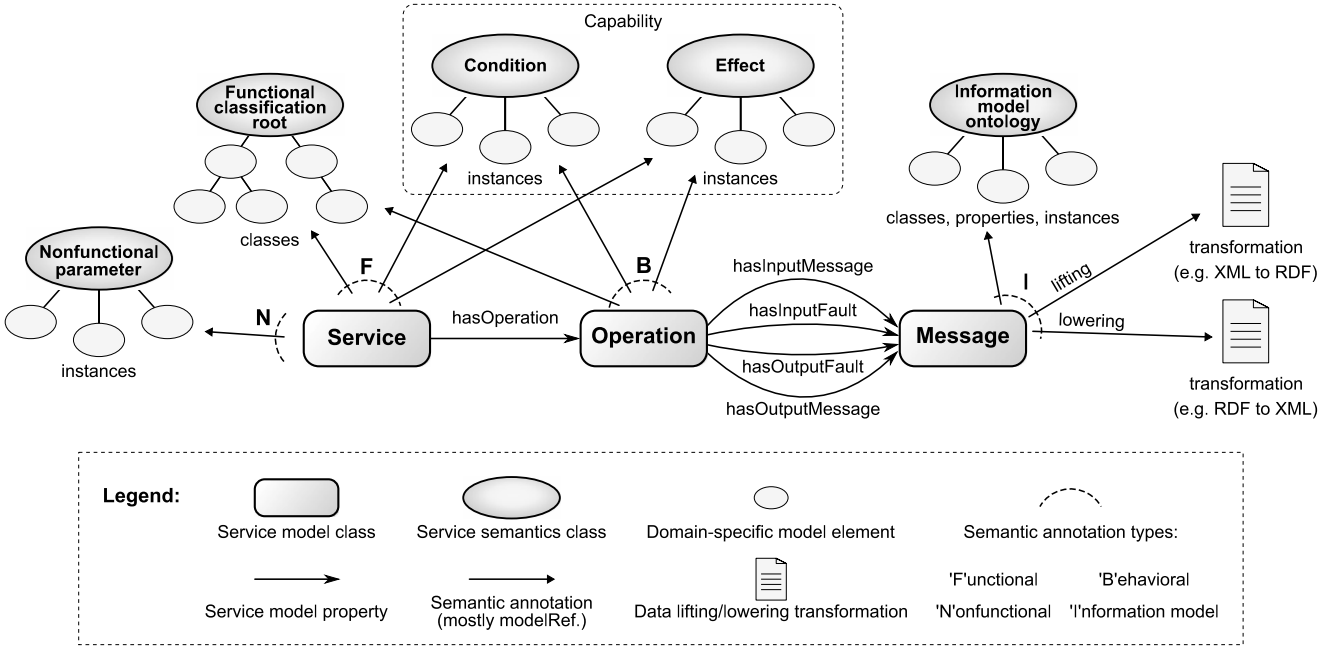


Fig. 6. The structure and use of the WSMO-Lite service ontology, annotating the service model from Fig. 3(b).

and the third column specifies what value the annotation can take. The fourth column (Context) shows where the value comes from, using the definitions from Section 4.1, and finally, the fifth column (Type) shows which of the four types of semantics this annotation describes: Functional, Nonfunctional, Behavioral and Information semantics.

The values of the lifting and lowering schema mapping properties are formalized as $f(data) \rightarrow X$ and $g(X) \rightarrow data$, where $X \subseteq E$ represents a set of data instances of some information model ontology Ω^I. The functions f and g are transformations between sets of ontological instances (X) and their representations in the underlying technical format, denoted as *data*. We do not constrain the underlying technical formats and the form of the transformation functions, therefore, the terms f, g and *data* are not formalized any further.

Fig. 6 illustrates the WSMO-Lite annotations in relation to the service model in a graphical form. The centrally-located components of the service model are annotated with pointers to domain-specific semantic descriptions that fit the service semantics classes defined in WSMO-Lite. The figure also illustrates how WSMO-Lite follows the design principle of *modularity*.

4.3. Using WSMO-Lite in WSDL and SAWSDL

The service model presented in Section 3, based on WSDL, is intentionally a simplification. Above, we defined how the minimal service model can be annotated with WSMO-Lite semantics, and here we show how the annotations should apply to the more complex model of WSDL.

Table 2 defines the WSMO-Lite semantic annotations for the minimal service model. Table 3 presents a concise summary of how the annotations apply in WSDL. The first column shows the components of the WSMO-Lite minimal service model, the second column indicates the type of semantics (functional, nonfunctional, behavioral and information-model) attached to the particular component, and the third column enumerates the corresponding WSDL components where the annotations belong.

Functional annotations. Most directly, the functional semantics of a service can be described with annotations on the WSDL *service* components. However, a major part of a WSDL description is the service *interface*, which "describes a Web service in terms of the messages it sends and receives", doing it "by grouping related messages into operations" [4].

Table 3
WSMO-Lite semantics in WSDL.

WSMO-Lite svc. model	Type	WSDL component
Service	F	Service or interface
Service	N	Service
Operation	B	Interface operation
Message	I	Element declaration or type definition

From the point of view of service semantics, WSDL makes no assertions about different services that implement the same interface, only that they will accept and emit messages with the structure defined in the interface operations, and sequenced according to the operation message exchange patterns. In other words, the WSDL specification does not mandate that an interface should be tied to any particular functionality that can be achieved using its operations.

Still, if a WSDL interface is created to support certain functionality, the description of this functionality can be added as a semantic annotation on the WSDL interface, so that it applies to all WSDL services that use this interface. It is also possible that both the service and its interface are annotated with functional descriptions, when the service restricts the functionality of the interface. Furthermore, in WSDL 2.0, interfaces may *extend* other interfaces. The functionality of an interface then includes the functionalities of the interfaces extended by it.

Nonfunctional annotations. Nonfunctional properties define incidental details and policies specific to the implementation or running environment of a service; therefore, they are naturally expressed as annotations of the WSDL service component.

Note that while the SAWSDL specification only describes the use of *model reference* annotations on WSDL *interface* components (along with some of their subcomponents, such as *operations*) and on XML Schema *element declaration* and *type definition* components, it allows the annotation of all the other components in WSDL, including *service*. Our use of *model references* with functional and nonfunctional annotations on *service* components is fully within the spirit of SAWSDL.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

Behavioral annotations. WSMO-Lite does not define a specific construct for behavioral descriptions; instead, we use functional annotations of service operations and we expect that the client will be able to decide the ordering of operation invocations based on what the operations do.[13]

In WSDL, service operations are described in the service interface; the intended behavior of an interface can be described with annotations on its operation components.

Notably, while WSDL focuses on the messaging and networking aspects of a service description, the use of HTTP has necessitated that WSDL introduce a semantic flag for marking operations as *safe* as defined in the Web architecture [10]. We discuss this property some more in Section 5.2; here it will suffice to say that operation safety is a part of the behavioral semantics of a service. An operation marked in WSDL as *safe* should be viewed in WSMO-Lite as having a *model reference* annotation with the value http://www.w3.org/ns/wsdl-extensions#SafeInteraction, which is treated as a functional category of safe interactions.

Information model annotations. The semantics of the exchanged data is expressed through annotations on the message schemas. A *model reference* annotation on an XML Schema element declaration or type definition points to a description of the semantics of the data described by the schema. For instance, an annotation pointing to an ontology class or relation means that the data will define an instance (or multiple instances) of that class/relation.

As few Web services use RDF-based messages [31], semantic annotations of operation inputs and outputs should be accompanied by pointers to the appropriate lowering and lifting transformations, also on the XML Schema element declarations or type definitions. A lifting transformation should accept documents valid according to the schema, and produce the equivalent RDF data. A lowering transformation takes RDF data as its input, and should produce an XML document that is valid according to the schema.

For lifting and lowering with XML-based Web services, we recommend the use of the language XSPARQL,[14] a fusion of SPARQL and XQuery, a query and transformation language able to process RDF and XML data sources and return RDF or XML. The W3C Member Submission [32], which defines the language, also includes a use case that demonstrates the use of this language for SWS lifting and lowering.

5. hRESTS and MicroWSMO: annotating RESTful Web services

As shown in Section 3, the minimal WSMO-Lite service model, which is a simplification of WSDL, also applies to RESTful services. However, WSDL is not commonly accepted for describing RESTful services. Alternative REST-oriented service description languages have been proposed; most prominent among these is the Web Application Description Language WADL [15]. WADL describes RESTful services as sets of interlinked resources, but effectively it has the same function as WSDL—to be the basis for tool support.

Despite the existence of WADL (and WSDL 2.0, the REST-friendly version of WSDL), the providers of RESTful services have so far been reluctant to provide any kind of machine-readable service descriptions [31]. Instead, RESTful services are commonly described in textual documentation, most often in the form of HTML pages.[15] The lack of machine-processible descriptions limits the possible tool support for users of RESTful services, including any semantic automation.

Since most RESTful APIs are described in HTML documentation, we propose here two *microformats*, hRESTS (which stands for "HTML for RESTful Services"), and MicroWSMO ("Microformat for WSMO-Lite"[16]), which together provide a simple mechanism for annotating HTML service documentation so that it becomes machine-processible and amenable to adding semantics.

hRESTS and MicroWSMO are two separate microformats because like WSDL, hRESTS is also useful by itself, without semantic annotations. hRESTS was developed first, in international collaboration (cf. [13]), as the common ground for facet-browsing of Web APIs in SA-REST [33], and for semantic automation with MicroWSMO. Modeling hRESTS and MicroWSMO as microformats for HTML documentation follows our design principles of *proximity to underlying standards* and *lightweight minimality*, and modeling them as two separate microformats stems from the principles of *modularity*.

Microformats are an "adaptation of semantic XHTML that makes it easier to publish, index, and extract semi-structured information" [14], an approach for annotating human-oriented Web pages so that key information is machine-readable. A microformat is mainly a collection of keywords that are used as *class*[17] names on HTML elements to indicate the type of data contained by the elements, and keywords used on hyperlinks to specify relations between resources. An HTML page with microformat annotations works in a Web browser as any other HTML page, while also allowing programmatic extraction of the contained data, regardless of the presentation structure of the page. There are already microformats for contact information, calendar events, etc., supported by a variety of tools.[18]

Alternatively to introducing a microformat to annotate the HTML documentation of RESTful services, we can also use RDFa [34], which provides a general syntax for embedding RDF data in HTML. In comparison with RDFa, microformats are less intrusive in the underlying HTML and easier to author, but they require custom parsers, where RDFa parsers will extract any RDFa content from any HTML page.

This section first describes the microformats and then turns to RDFa. In Section 5.1 we define the microformat hRESTS and its RDF interpretation, and in Section 5.2 we specify the microformat MicroWSMO. Section 5.3 discusses the straightforward application of RDFa as an alternative to using the microformats.

5.1. hRESTS: a service description microformat

As already stated, public Web services come with HTML documentation. Textual documentation in HTML is the prevalent form of Web API descriptions, and in many cases it is the only one.

Typically, such documentation will list the available operations (under various names such as *API calls*, *methods*, *commands* etc.), their URIs and parameters, the expected output data, any error conditions and so on; it is, after all, intended as the documentation of a programmatic interface. In effect, the documentation of RESTful services is usually closer to the structure of WSDL (a

[13] In [12], we have shown that the WSMO-Lite behavioral semantics (functional annotations on service operations) can be translated into a WSMO *choreography* (cf. [3]), which is an explicit behavioral description.

[14] http://xsparql.deri.org/.

[15] For example, see docs.amazonwebservices.com/AmazonSimpleDB/2007-11-07/DeveloperGuide and flickr.com/services/api.

[16] MicroWSMO is so named for historical reasons; a more direct name "SA-hRESTS" (Semantic Annotations for hRESTS) would be confusingly close to SA-REST by [33]; and another alternative, "MicroSAWSDL", would imply close ties with WSDL, which would be undesirable with RESTful services.

[17] Most HTML elements may have an attribute class that can carry a list of arbitrary strings. Classes are commonly used for assigning visual styles to elements, but they can also be used as annotations about the meaning of the elements' contents.

[18] See microformats.org for examples and further information about microformats.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

set of operations) than that of WADL (a set of data resources). In Section 3.3, we have explained why the WSDL-based simple service model is not really foreign to RESTful Web services. This justifies why our microformat hRESTS straightforwardly follows the WSMO-Lite minimal service model, rather than having a resource-oriented structure like WADL.

The following is an example of a typical operation description, encoded in HTML:

Operation description	HTML source
ACME Hotels service API	ACME Hotels service API
Operation getHotelDetails	Operation getHotelDetails
Invoked using the method GET at `http://example.com/h/{id}`	Invoked using the method GET at <code> `http://example.com/h/{id}` </code>
Parameter:	Parameter:
`id`- the identifier of the particular hotel	<code> id</code>—the identifier of the particular hotel
Output value:	Output value:
hotel details in an `ex:hotelInformation` document	hotel details in an <code> ex:hotelInformation</code> document

Such documentation has all the details necessary for a human to be able to create a client program that can use the service. In order to tease out the technical details (operations, addresses, HTTP methods, input and output data formats), the HTML documentation needs to be amended in some way, which is what the microformat hRESTS does. hRESTS adds machine-readable structure to HTML service descriptions and it additionally identifies two key pieces of information about an operation: the *address* (URI template) and the *HTTP method* used by the operation. In effect, hRESTS is analogous to WSDL, albeit less detailed.

The microformat is made up of a number of HTML classes that correspond to the various parts of our service model and their properties:

- `service` specifies the description of the whole service,
- `operation` marks a single operation within a service,
- `address` defines the URI template for an operation,
- `method` defines the HTTP method for an operation,
- `input` marks a block describing the inputs of an operation,
- `output` does the same for the outputs, and
- `label` specifies a human-readable name of a service or of an operation.

The classes `service`, `operation`, `input` and `output` are used on HTML *block markup*, marking a block of the HTML page that describes a whole service, a particular operation or one of its messages. In contrast, the classes `address`, `method` and `label` are used on *textual markup*, indicating that the text content of the marked element is the actual address URI template, the method name or the label.

Note that both the classes `address` and `method`, naturally nested in `operation`, may also be specified on the level of `service`, in which case these values serve as defaults for operations that do not specify them. In the absence of any explicit value for method, the default is GET.

The `input` and `output` classes serve as extension points—hRESTS does not provide for further machine-readable information about the inputs and outputs. In Section 5.2, MicroWSMO adds semantic annotations in these blocks; other extensions may be developed in the future for example to specify formal data schemas for the messages.

Listing 1 shows hRESTS annotations (in bold) in the HTML code of the sample service description shown earlier. Note the added `<div>` and `<span>` blocks that add nested element structure to the description; they do not otherwise affect the presentation of the HTML documentation.

```
1  <div class="service" id="svc">
2   <h1><span class="label">ACME Hotels</span> service API</h1>
3   <div class="operation" id="op1">
4    <h2>Operation <span class="label">getHotelDetails</span></h2>
5    <p> Invoked using the <span class="method">GET</span>
6     at <code class="address">http://example.com/h/{id}</code><br/>
7     <span class="input">
8      <strong>Parameters:</strong>
9      <code>id</code> — the identifier of the particular hotel
10    </span><br/>
11    <span class="output">
12     <strong>Output value:</strong> hotel details in an
13     <code>ex:hotelInformation</code> document
14    </span>
15   </p>
16  </div></div>
```

Listing 1: Example hRESTS service description

For semantic automation, hRESTS descriptions are not meant to be processed directly in the HTML form, instead we map them to RDF data that follows the service model from Section 3. This way, WSMO-Lite tools need not distinguish between WSDL services and RESTful APIs—except during actual service invocation, they can be treated as equals. In order to represent the operation address and method in the RDF form, we add two RDF properties, hr:hasMethod[19] and hr:hasAddress. If defaults for method and address are specified on the `service` level, the RDF form reflects the default values already applied, that is, instances of wl:Service will never have either hr:hasMethod or hr:hasAddress, while instances of wl:Operation should always have both.

The mapping of hRESTS descriptions into RDF can be carried out through GRDDL [35], a mechanism for extracting RDF data from Web pages, particularly suitable for processing microformats. With GRDDL, the Web page is processed by one or more XSLT transformations that result in RDF triples; the result is an RDF view on the content of the page.

The following subsection contains an example of HTML annotated with both microformats and it shows the RDF data generated from such HTML.

5.2. MicroWSMO semantic annotations for hRESTS

Building on hRESTS, we can now proceed to annotate descriptions of RESTful services with semantics. Since hRESTS is by design similar to WSDL, we add semantic annotations using the standard SAWSDL properties, adapted to a microformat syntax in a microformat called MicroWSMO. Similarly to how SAWSDL is a layer for semantic annotations of WSDL (the machine-readable service description language with support in development tools), also MicroWSMO is a layer for semantic annotations of hRESTS (the service description microformat that aims to provide for development tool support).

SAWSDL annotations are URIs that identify semantic concepts and data transformations. The annotation URIs can be added to the HTML documentation of RESTful services in the form of hypertext links. HTML [36] defines a mechanism for indicating the relation represented by a hyperlink; the relation is specified in the `rel` attribute. Along with `class`, the `rel` attribute is also commonly used by microformats.

In accordance with SAWSDL, MicroWSMO consists of the following three types of link relations:

- `model` indicates that the link is a model reference,
- `lifting` and `lowering` then denote links to the respective data transformations.

[19] The namespace prefix hr stands for http://www.wsmo.org/ns/hrests#.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

The `model` link relation, on a hyperlink present within an hRESTS `service`, `operation`, `input` or `output` block, specifies that the link is a model reference from the respective component to its semantic description. We can directly apply WSMO-Lite semantics here, as discussed in Section 4.2 and summarized in Table 2.

Note that HTTP [9] defines some limited semantics for the various methods. In particular, the method GET is supposed to be *safe*, and the methods PUT and DELETE are *idempotent*. The safety property of GET is of particular interest, allowing opportunistic client behavior such as pre-caching and Web crawling, or even automated offer discovery, which we describe in [37]. To include operation safety in the service behavioral semantics, MicroWSMO can automatically add a model reference to wsdlx:SafeInteraction[20] on any operation that uses GET.[21]

The `lifting` and `lowering` link relations, on hyperlinks present within hRESTS `input` or `output` blocks, specify that the links point to the respective lifting and lowering schema mapping data transformations between the knowledge representation format of the service's data ontology and the network transmission syntax of the messages of the service.

In contrast to data lifting and lowering for WS–∗ services, implementing lifting and lowering for RESTful services is complicated by the following factors: (a) XML is not as prevalent in RESTful services, so data transformations may need to support other formats, such as JSON[22]; (b) the input data to a RESTful service operation is not necessarily in a single document, as parts of the input data may instead be URI (query) parameters; and (c) the meaning of the response data depends also on the response status code (in case of faults).

Listing 2 illustrates the use of the MicroWSMO link relations on semantic annotations added to the hRESTS description from Listing 1. The `model` link relation is used on lines 4 and 13. Line 4 specifies that the service supports hotel reservations (the URI would identify a category in some classification of services), and line 13 defines the input of the operation to be an instance of the class Hotel, which is a part of the service's data ontology. Line 15 shows a link to a lowering transformation that would presumably map a given instance of the class Hotel into the ID that the service expects as a URI parameter.

```
1  <div class="service" id="svc">
2    <h1><span class="label">ACME Hotels</span> service API</h1>
3      <p>This service is a
4        <a rel="model" href="http://ecommerce.example/hotelReservation">
5          hotel reservation</a> service.
6      </p>
7    <div class="operation" id="op1">
8      <h2>Operation <span class="label">getHotelDetails</span></h2>
9      <p> Invoked using the <span class="method">GET</span>
10       at <code class="address">http://example.com/h/{id}</code><br/>
11       <span class="input">
12         <strong>Parameters:</strong>
13         <a rel="model" href="http://example.com/data/onto.owl#Hotel">
14           <code>id</code></a> — the identifier of the particular hotel
15           (<a rel="lowering" href="http://example.com/data/hotelID.xsparql">
16            lowering</a>)
17       </span><br/>
18       <span class="output">
19         <strong>Output value:</strong> hotel details in an
20         <code>ex:hotelInformation</code> document
21       </span>
22     </p>
23   </div></div>
```

Listing 2: Example MicroWSMO semantic description

To demonstrate the RDF mapping of both hRESTS and MicroWSMO, Listing 3 shows the RDF data that can be extracted from the example description in Listing 2, for instance by using the GRDDL XSLT transformation mentioned in the preceding subsection. The result has the same structure as the RDF data obtained from WS–∗ descriptions in WSDL and SAWSDL. Using this RDF form of the service descriptions, we can treat RESTful services described with hRESTS and MicroWSMO just like WS–∗ services described with WSDL and SAWSDL.

```
1  @prefix ex: <http://example.com/serviceDescription.html#> .
2  @prefix hr: <http://www.wsmo.org/ns/hrests#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
4  @prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
5  @prefix wsdlx: <http://www.w3.org/ns/wsdl−extensions#> .
6
7  ex:svc a wl:Service ;
8    rdfs:isDefinedBy <http://example.com/serviceDescription.html> ;
9    rdfs:label "ACME Hotels" ;
10   sawsdl:modelReference <http://ecommerce.example/hotelReservation> ;
11   hr:hasOperation ex:op1 .
12 ex:op1 a wl:Operation ;
13   rdfs:label "getHotelDetails" ;
14   hr:hasMethod "GET" ;
15   sawsdl:modelReference wsdlx:SafeInteraction .
16   hr:hasAddress "http://example.com/h/{id}"^^hr:URITemplate ;
17   wl:hasInputMessage [
18     a wl:Message ;
19     sawsdl:modelReference <http://example.com/data/onto.owl#Hotel> ;
20     sawsdl:loweringSchemaMapping <http://example.com/data/hotelID.xsparql>
21   ] ;
22   wl:hasOutputMessage [
23     a wl:Message ;
24   ] .
```

Listing 3: RDF data extracted from Listing 2

5.3. RDFa: an alternative to hRESTS and MicroWSMO

Alternatively to using microformats to capture the service model structure in the HTML documentation of RESTful Web services, we can also employ RDFa [34], directly using the RDF-based WSMO-Lite service model. RDFa specifies a collection of generic XML attributes for expressing RDF data in any markup language, and especially in HTML.

In our case, the difference between the use of a microformat or RDFa boils down to several considerations:

- the microformat syntax is simpler and more compact than RDFa;
- RDFa represents the full concept URIs and thus facilitates the coexistence of multiple data vocabularies in a single document, where microformats may run into naming conflicts;
- processing microformats requires vocabulary-specific parsers (such as our XSLT transformation described in Section 5.1), while parsing the RDF data from RDFa is independent from any actual data vocabularies.
- RDFa cannot support language-specific shortcuts (such as the defaulting of address and method properties from a service to its operations), but with tool support, users may not need such shortcuts.

Mindful of the size of the article, we illustrate the RDFa form of the WSMO-Lite service model with SAWSDL annotations (instead of the hRESTS/MicroWSMO microformats) with a brief snippet in Listing 4.

[20] wsdlx:SafeInteraction is a class defined by the WSDL RDF mapping [25] to represent the safety property.

[21] A low but significant number of Web applications use HTTP GET to perform side effects, such as deleting an item from a container or confirming mailing list unsubscription. Web search engine crawlers and Web accelerator programs have been using server-provided restrictions (`robots.txt`) and heuristics (such as the presence of URI parameters after '?') to guess whether GET will be safe on a given URI, limiting their reach to avoid unintended consequences for broken Web applications. Such heuristics can also be built into a MicroWSMO parser so that the wsdlx:SafeInteraction model reference would only be attached to operations whose address is judged as safe.

[22] http://www.json.org/.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

```
<div typeof="wl:Service" about="#svc"
      xmlns:wl="http://www.wsmo.org/ns/wsmo−lite#"
      xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf−schema#">
  <span rel="rdfs:isDefinedBy" resource="" />
  <h1><span property="rdfs:label">ACME Hotels</span> service API</h1>
   <p>This service is a
    <a rel="sawsdl:modelReference"
       href="http://ecommerce.example/hotelReservation">
      hotel reservation</a> service.
   </p>
  <div rel="wl:hasOperation"><div typeof="wl:Operation" about="#op1">
    <h2>Operation <code property="rdfs:label">getHotelDetails</code></h2>
```

Listing 4: Example service description with RDFa annotations

6. Illustrative SWS automation algorithms

The preceding sections show how Web service descriptions can be annotated with semantics, and the ontology in which the semantics are expressed. Semantic descriptions are intended to support tasks such as service discovery and composition, therefore in this section we show several algorithms for these tasks, in order to evaluate that the proposed languages can actually support Web service automation. Adapted from existing literature, the presented algorithms are not necessarily meant to be the most powerful or the most efficient ones, instead they are meant to demonstrate the versatility of our lightweight semantic descriptions.

The process of using Web services can be split into the following tasks: discovery, negotiation, ranking and selection, composition, mediation and invocation. Here, we focus on algorithms for Web service discovery. We do not intend to imply that the presented algorithms are the most efficient or the most user-friendly; they are a selection of common and proven approaches, on which we can easily demonstrate how such algorithms can be adapted to WSMO-Lite. By adopting known and tested approaches, we show how SWS research may converge on a technology like WSMO-Lite that is close to Web services practitioners. In other words, with WSMO-Lite we do not have to discard the existing body of research on SWS automation algorithms.

Adapting existing algorithms to WSMO-Lite involves the reconciliation of terminology and the refinement of WSMO-Lite semantics. WSMO-Lite has two effects on terminology that have to be accepted in an adapted algorithm: first, WSMO-Lite takes the SAWSDL point of view building bottom-up on the underlying technical descriptions, as opposed to top-down from a semantic model; and second, WSMO-Lite distinguishes between the four types of semantics (functional, nonfunctional, behavioral and information) and it specifies four RDFS classes to express them.

Since WSMO-Lite semantics are intentionally limited, adapting a SWS automation algorithm may involve filling in concrete details (e.g. instantiation of classes) about WSMO-Lite semantics that are used by the algorithm, effectively refining the semantics defined by WSMO-Lite. Additionally, an algorithm must also define what kinds of data it requires from the user to specify a *goal*.

The following subsections present selected algorithms adapted to WSMO-Lite, demonstrating both the use of WSMO-Lite terminology and the refinement of its semantics. We have chosen to focus on algorithms that use functional semantics, as these are non-trivial; adapting algorithms that use information and nonfunctional semantics is straightforward, and working with behavioral semantics in WSMO-Lite is analogous to functional semantics.

6.1. Functional Web service discovery

The scope of the term "discovery" can be understood very widely, encompassing a Web crawler that finds existing service descriptions, a matchmaker that selects known services that match a given user goal, and even a negotiation step that finds the concrete offers of the selected services. In this section, we restrict the meaning of discovery to functional Web service matchmaking; i.e., the client's goal specifies *what* a desired service should be able to do, and the matchmaker compares it to the advertised functionalities of the known Web services.

Klusch [38] presents the most recent survey of semantic service discovery approaches. Along with logic-based matchmakers, he also investigates non-logic-based mechanisms that employ measures such as text similarity to determine the degree of match between a service and a user request. Among the logic-based approaches, Klusch further investigates what kinds and parts of service semantics are considered for matching, especially pointing out how various approaches use different combinations of the descriptions of service inputs, outputs, preconditions and effects (together known as IOPE).

Among full *IOPE* matchmakers, which use all the four semantic aspects, Klusch cites the work of [39],[23] which we adopt here for the purpose of demonstrating matchmaking with WSMO-Lite. The approach of Keller et al. is based on a simple concept of modeling Web services and goals as sets of relevant objects: a service can deliver certain objects, and a goal requests them. We use Keller et al.'s naming of these sets: a Web service $\mathcal{W}$ is represented through a set named $R_{\mathcal{W}}$, and a goal $\mathcal{G}$ is represented through a set named $R_{\mathcal{G}}$.

Keller et al. distinguish two *modeling intentions*: existential and universal. In *existential* modeling, the sets overdescribe the services and goals—a goal $\mathcal{G}$ will be satisfied if any object(s) from $R_{\mathcal{G}}$ is delivered; and a service $\mathcal{W}$ can only really deliver some objects from $R_{\mathcal{W}}$. For example, a goal set can describe all room reservations in a given city at given dates, but the intention is to get only one reservation; and a service set can describe all the possible room reservations at a given hotel, but only some rooms will be available at some dates. In *universal* modeling, the sets describe the service or the goal exactly: the goal requires all the objects from $R_{\mathcal{G}}$ to be delivered, and the service can deliver all the objects from $R_{\mathcal{W}}$. While universal modeling is potentially more accurate, Keller et al. list no plausible use cases where the required level of description detail would be desirable or practical. Therefore, we restrict our discussion only to the existential modeling intention.

In order for $\mathcal{W}$ and $\mathcal{G}$ to be considered a match, the sets $R_{\mathcal{W}}$ and $R_{\mathcal{G}}$ have to be interrelated. There are four possible set-theoretic relations, with an inherent match-degree ranking among them (the list goes from the best match to the worst):

- *Exact match of equal sets* ($R_{\mathcal{G}} = R_{\mathcal{W}}$): the service may be able to deliver all the objects requested by the goal, and it cannot deliver any other, irrelevant objects. This is the closest match between a goal and a service. For example, a goal requests accommodation in Rome, and there is a service that specializes on accommodation in Rome.
- *Web service subset of goal* ($R_{\mathcal{G}} \supseteq R_{\mathcal{W}}$): the service can only deliver some of the objects requested by the goal; it cannot deliver irrelevant objects. For example, if the goal requests accommodation in Rome, a particular service may only cover budget hotels in this city. The service description indicates that the service has limitations with respect to the goal (only budget hotels), but it can be acceptable to the client (whose goal does not specify the client's demands on service level or price).
- *Goal subset of Web service* ($R_{\mathcal{G}} \subseteq R_{\mathcal{W}}$): the service may be able to deliver all objects requested by the goal, but it may also (or even

[23] We use a newer publication by Keller et al. which superseded that cited by Klusch.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

only) deliver irrelevant objects. For example, a service offering accommodation in Italy in general may have a limited coverage of hotels in Rome. In this case we merely have a possible match.
- *Nonempty intersection of service and goal* ($R_{\mathcal{G}} \cap R_{\mathcal{W}} \neq \emptyset$): the service may be able to deliver some of the requested objects, but it may also deliver irrelevant objects. For example, a Hilton hotel reservation service is clearly limited with respect to the hotel (it only books Hilton hotels), but it still is a possible match because the description does not say whether there is a Hilton hotel in Rome.

The four match degrees may be used to rank the discovered services. For example, performing negotiation with better-matching services first can quicker lead to useful results, which the semantic client system can asynchronously display to the user.

WSMO-Lite provides two distinct mechanisms for describing the functionality of Web services: the lightweight but coarse-grained *functional classification*[24] that deals with taxonomies of functionality, and the more expressive *capability* described with preconditions and effects. Both types can be interpreted as descriptions of sets of objects. [39] only deal with capabilities; we extend their approach to discovery using functional classifications.

In the subsections below, we detail the concrete matchmaking algorithms that follow the set-based approach of Keller et al.: in Section 6.1.1, we discuss the extension of the work of Keller et al. for matchmaking with functional classifications, in Section 6.1.2, we summarize matchmaking with capabilities from [39], and then in Section 6.1.3 we combine the two separate approaches.

6.1.1. Matchmaking with WSMO-Lite functional classifications

Functional classification is the simpler one of the two mechanisms WSMO-Lite provides for functional description of Web services. Using SAWSDL model references, a Web service s can be associated with one or multiple categories $(c_1^s, \ldots, c_n^s)$ from one or multiple classification ontologies. For the purpose of the discovery algorithms here, we interpret those categories as specifying the sets of objects that can be delivered by Web services, as discussed above. Keller et al. do not consider functional classifications in their work. By interpreting functional categories as sets of objects, we devise here a straightforward extension of the approach of [39].

Multiple categories associated with one service are treated in conjunction—a service belongs to all the functionality categories with which it is associated. In other words, the service is described by the intersection of the functionality categories. Effectively, we can say the service is associated with a single functional category $R_{\mathcal{W}}$:

$$R_{\mathcal{W}} = \bigcap_{i=1\ldots n} c_i^s.$$

For selecting Web services, a user goal must specify a category of interest, $R_{\mathcal{G}}$, so that the matchmaking algorithm can return services associated with matching categories that are related with the goal category through the subclass relationships that make up the functionality classifications. For symmetry and simplicity, we describe the goal category also as the intersection of one or more given categories $(c_1^g, \ldots, c_m^g)$:

$$R_{\mathcal{G}} = \bigcap_{i=1\ldots m} c_i^g.$$

Since WSMO-Lite functionality classifications are built using the subclass relationship ($c_i \subseteq c_j$), we are limited to inspecting the subclass hierarchies to determine the match degrees between goals and Web services:

- $R_{\mathcal{G}} = R_{\mathcal{W}}$: the service exactly matches the goal if it is both a subset and a super-set of the goal, as defined below.[25]
- $R_{\mathcal{G}} \supseteq R_{\mathcal{W}}$: the service is a subset of the goal if the service is associated with a subcategory of each of the goal categories.
- $R_{\mathcal{G}} \subseteq R_{\mathcal{W}}$: the goal is a subset of the service if the goal is associated with a subcategory of each of the service categories.
- $R_{\mathcal{G}} \cap R_{\mathcal{W}} \neq \emptyset$: the service intersects the goal if any of the above is true, but also if we can find any one category that is a subcategory of all of the categories associated with both the goal and the service.[26]

Algorithm: matchmaker for WSMO-Lite functional classifications
Inputs: set S of known services annotated with funct. categories, goal categories $c_1^g, \ldots, c_m^g$
Result: set of tuples $\langle$*service, match degree from* $\{=, \supseteq, \subseteq, \cap\}\rangle$

```
M := ∅        (result set, initially empty)
for each s ∈ S
    δ := "–"        (match degree, initially "–" for "no match")
    if ∀i ∈ {1...n_s} ∃j ∈ {1...m_s} : c_j^g ⊆ c_i^s
        δ := "⊆"
    if ∀j ∈ {1...m_s} ∃i ∈ {1...n_s} : c_j^g ⊇ c_i^s
        if δ = "⊆"
            δ := "="
        else
            δ := "⊇"
    if ∃c : (∀i ∈ {1...n_s} : c ⊆ c_i^s) ∧ (∀j ∈ {1...m_s} : c ⊆ c_j^g)
        δ := "∩"
    if δ ≠ "–"
        M := M ∪ {⟨s, δ⟩}
return M
```

Fig. 7. Matchmaker algorithm for WSMO-Lite functional classifications.

Fig. 7 shows the matchmaking algorithm that embodies our adaptation of the work of Keller et al. to WSMO-Lite functional classifications; it formalizes the conditions for each of the match degrees.

The algorithm employs subsumption reasoning, which can in some languages (such as RDFS) be reduced to the well-known and tractable problem of graph reachability. In other words, discovery with functional classifications can provide good performance at the cost of expressivity—the level of detail practically possible in service and goal descriptions.

6.1.2. Matchmaking with WSMO-Lite preconditions and effects

In addition to the coarse-grained functionality classifications discussed above, WSMO-Lite supports fine-grained service description with logical expressions that capture the precondition and effect (together, the *capability*) of the service. Preconditions and effects can also be used to model services as sets of objects, as shown by [40] (referenced from [39] as a concrete realization of set-based discovery in formal logics). Here, we provide a summary of their approach, adapted to the terminology of WSMO-Lite.

Keller et al. propose two ways of expressing the sets of objects that represent Web services and user goals:

Simple semantic descriptions: the sets $R_{\mathcal{W}}$ and $R_{\mathcal{G}}$ are defined using first-order formulas $\phi(x)$ and $\psi(x)$[27] with one free variable each:

$\mathcal{W} : R_{\mathcal{W}} = \{x \mid \phi(x)\}$

$\mathcal{G} : R_{\mathcal{G}} = \{x \mid \psi(x)\}$.

[24] Creating taxonomies is an expensive, consensus-driven task with benefits diminishing as the number of categories grows, therefore we expect functionality classifications to be coarse-grained.

[25] Note that this is a one-way implication, not an exclusive *iff*: for example when using two overlapping classifications for which we do not have a formal mapping, it is possible that the service category is an exact match of the goal category but they use different terms so the matchmaker has no means of verifying the match.

[26] We assume here that any explicitly defined functionality category can be seen as a *non-empty* set of objects that some service can deliver.

[27] Note that in contrast to [40], we swap the use of the symbols ϕ and ψ, for consistency with preceding sections.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hREST: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

Table 4
Proof obligations for the four types of set-theoretic relationship between the goal and the service object sets.
Source: From [40].

Set relationship	Proof obligation
Simple semantic descriptions	
$R_{\mathcal{G}} = R_{\mathcal{W}}$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x : (\psi(x) \leftrightarrow \phi(x))$
$R_{\mathcal{G}} \subseteq R_{\mathcal{W}}$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x : (\psi(x) \rightarrow \phi(x))$
$R_{\mathcal{G}} \supseteq R_{\mathcal{W}}$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \forall x : (\psi(x) \leftarrow \phi(x))$
$R_{\mathcal{G}} \cap R_{\mathcal{W}} \neq \emptyset$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists x : (\psi(x) \wedge \phi(x))$
Rich semantic descriptions	
$R_{\mathcal{G}} = R_{\mathcal{W}_{\mathcal{G}}}$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n : (\forall x : (\psi(x) \leftrightarrow \phi(x, i_1 \ldots i_n)))$
$R_{\mathcal{G}} \subseteq R_{\mathcal{W}_{\mathcal{G}}}$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n : (\forall x : (\psi(x) \rightarrow \phi(x, i_1 \ldots i_n)))$
$R_{\mathcal{G}} \supseteq R_{\mathcal{W}_{\mathcal{G}}}$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n : (\forall x : (\psi(x) \leftarrow \phi(x, i_1 \ldots i_n)))$
$R_{\mathcal{G}} \cap R_{\mathcal{W}_{\mathcal{G}}} \neq \emptyset$	$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \exists i_1, \ldots, i_n : (\exists x : (\psi(x) \wedge \phi(x, i_1 \ldots i_n)))$

Rich semantic descriptions: the above is extended with a notion of *input data* that influences the set of objects delivered by the service. The service description expresses a precondition $\phi^{pre}(i_1 \ldots i_n)$ on the inputs, and an effect $\phi^{eff}(x, i_1 \ldots i_n)$, combined together in $\phi(x, i_1 \ldots i_n)$. Thus, the set $R_{\mathcal{W}}$ depends on the concrete input data, which is captured in the goal description as a set of instances $D_{\mathcal{G}}$. The service is not considered to deliver anything meaningful if the precondition cannot be fulfilled by a given set of concrete inputs.

$$\begin{aligned} \mathcal{W}: \quad & R_{\mathcal{W}} = \{x \mid \exists i_1 \ldots i_n \in D_{\mathcal{G}} : \phi(x, i_1 \ldots i_n)\} \\ & \phi(x, i_1 \ldots i_n) \leftrightarrow \phi^{pre}(i_1 \ldots i_n) \wedge \phi^{eff}(x, i_1 \ldots i_n) \\ \mathcal{G}: \quad & R_{\mathcal{G}} = \{x \mid \psi(x)\} \\ & D_{\mathcal{G}} = \{i_1, \ldots, i_m\}. \end{aligned}$$

In an actual WSMO-Lite service description, the formula $\phi(x)$ or $\phi^{eff}(x, i_1 \ldots i_n)$ is captured in a suitable logical language such as RIF [28] as the *effect* of a Web service capability K (kappa, see Section 4.1), and the formula $\phi^{pre}(i_1 \ldots i_n)$ is similarly captured as the capability's *precondition*.

In order to evaluate the various types of match, an automated reasoner can be employed to prove logical relationships between the formulas. In Table 4, we summarize the proof obligations defined in [40]. All of the proof obligations are of the form

$$\mathcal{W}, \mathcal{G}, \mathcal{O} \models \textit{expression}$$

where $\mathcal{W}$ is the definition of the Web service, $\mathcal{G}$ is the definition of the goal, $\mathcal{O}$ is a set of ontologies to which both descriptions refer, and *expression* is the actual formula that combines $\psi(x)$ with $\phi(x)$ or $\phi(x, i_1 \ldots i_n)$.

A discovery algorithm straightforwardly follows from the table: the algorithm would simply use a reasoner to evaluate the proof obligations for the given goal and for each known service, returning the matching services along with their match degrees.

The rich semantic descriptions are backward-compatible with the simple semantic descriptions: if there is no precondition, and the effect is independent of the input data $i_1 \ldots i_n$, the proof obligations in the lower half of Table 4 reduce to those from the upper half of the table. This compatibility makes it possible for simple descriptions to be used together with rich descriptions in a single system. It also enables gradual adoption of semantic complexity—a system can first only support the simple descriptions, and later adopt rich descriptions when the service providers and users become familiar with the logic languages and with the ontologies involved in the system; the older simple descriptions will still be evaluated correctly.

6.1.3. Combining functional classifications and capabilities

So far, we have adapted the work of Keller et al. to WSMO-Lite preconditions and effects (Section 6.1.2), and we have extended it to deal with WSMO-Lite functionality classifications (Section 6.1.1). Here, we discuss how these two mechanisms can usefully be combined.

To combine functional classification discovery with capability discovery, functional classification descriptions can be straightforwardly translated to capability descriptions as follows:

$$\mathcal{W} : R_{\mathcal{W}} = \bigcap_{i=1\ldots n} c_i^s \Longrightarrow \phi(x) = (\forall i = 1 \ldots n : x \in c_i^s).$$

Equally, goal descriptions can also be translated in this manner. The translation would enable capability discovery to use functional classification annotations. However, due to the difference in nature of taxonomies used for functional classifications and ontologies used for fine-grained logical expressions, we do not expect that creating mappings between them would be economical, therefore the above translation would be unlikely to lead to new matches (better matchmaking recall).

Alternatively, the two approaches can be combined in an efficient two-stage discovery process, where functional category matching precedes precondition and effect evaluation. Functionality classifications can be expected to be coarse-grained, with each category expressing the consensus of a community. On the other hand, logical preconditions and effects provide the expressivity to describe services and goals in great detail, however, at the cost of decreased performance, due to the computational complexity of logical reasoning and proof. In combination, discovery over a large Web service registry can perform an efficient first step using functional classifications, and then evaluate the preconditions and effects only on the services with matching categories. Thus, WSMO-Lite offers improved scalability to large Web service registry sizes over existing approaches such as OWL-S and WSMO, where discovery traditionally always performs computationally intensive reasoning.

When used alone, capability descriptions need to be detailed and yet comprehensive to describe the service's functionality unambiguously. In combination with functionality classifications, a functional category specifies the broad functionality of the given service, therefore the preconditions and effects in the capability description need only express desired additional details. Effectively, the logical expressions that make up the capability descriptions may be simpler, easing the task of authoring the semantic service descriptions.

The two-stage combination of the two discovery approaches (based on functionality classifications and on formal capabilities) requires also that the user goal contain both kinds of data: a requested functionality category and a description of the desired effect (plus input data if rich capability descriptions are used). Also here, the logical expression that defines the desired effect is simplified because it operates in the context of the requested functionality category specified by the goal.

In summary, combining matchmaking through functionality classifications and through logical capabilities has benefits both in performance and in complexity of service descriptions.

Notably, Stollberg [41] achieves similar effects on discovery scalability and description complexity by enhancing WSMO discovery [39] with the use of goal templates and Web service templates. The templates are presumably defined through a similar process as our functionality classifications—by reaching consensus on useful subdivision of a domain. Goal templates can be matched against Web service templates ahead of the time of concrete discovery, so that a concrete goal need only be matched against the concrete Web services that are described using Web service templates that match the template of the goal.

As Stollberg's work focuses on the performance benefits from pre-matching of templates, it does not pursue the difference between authoring logical capability descriptions and creating hierarchies of functionality templates. WSMO-Lite separates functionality classifications from logical capabilities, stressing the differences in granularity and degree of collaboration expected to author the two types of semantic descriptions.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

6.2. Functional Web service composition

Web service composition is the process of combining existing services in such a way that they provide a new desired functionality; the result is also often called a service composition, or a composite service. There are many different strategies for service composition, as shown by the survey of [42].

Among other aspects, the survey emphasizes the distinction between *manual composition* and *automatic composition*. To support manual composition, research focuses on languages that describe compositions, and tools that help the user with composing selected services. For automated composition, research investigates languages for describing services, and algorithms that use service descriptions to compose services that together can achieve a specified goal. Semantic technologies such as WSMO-Lite are especially suited for supporting automated composition, which is therefore the scope of this section.

Different automated composition approaches use varying levels of detail in service descriptions. A recent example by [43] is representative of many approaches that match services using a sequence based on their inputs and outputs. Such approaches assume that the inputs and outputs of a service implicitly reflect the service's functionality. Conversely, more sophisticated approaches such as the one from Hoffmann et al. [44] use the preconditions and effects of Web services as *explicit* functional descriptions, decoupling message types from service functionality.

These two levels mentioned so far are commonly called *functional-level composition*, treating Web services as functions with single points of input and output. In contrast, *process-level composition* (e.g. [45]) takes into account the behavioral interfaces of the composed services, treating them as processes rather than atomic functions. Functional-level composition is tractable, but because it does not take into account the services' behavioral interfaces, the composition solutions are not guaranteed to be actually executable. Rather, they support the human designer with rich information about possible compositions. Ideally, only minor modifications should correct any remaining mismatches.

The result of Web service composition may simply be a linear sequence of services (e.g. [44]), or it can be a non-linear composition with parallel and/or conditional branches [43,45].

In this section, we adapt to WSMO-Lite the composition approach from Hoffmann et al. [44]. It illustrates powerful composition with service preconditions and effects without delving into the complexity of process-level composition. While the algorithm produces linear compositions, it is a property of the algorithm—WSMO-Lite can just as well support other composition approaches.

The composition algorithm of Hoffmann et al. is defined using a formalism that is independent of the underlying SWS technology; here we show how it can be used with WSMO-Lite. For this purpose, we summarize the approach in enough detail to make the section self-contained.

The formalism used by Hoffmann et al. follows Winslett's [46] *possible models* approach to define the semantics of updates that occur when a Web service is applied to a given expected state. In order to explain the high-level functioning of the algorithm, we show here a subset of the formalism.

In the formalism, Web services are represented with their preconditions and effects. In WSMO-Lite, these are captured as a capability K (kappa, see Section 4.1). Further, user goals are defined by Hoffmann et al. through preconditions and effects, where the precondition serves only as the supply of initial constants. Therefore, we capture a goal directly as the effect and a set of the initial constants:

$$\begin{aligned} \mathcal{W}: &\quad K = (\Sigma, \phi^{pre}, \phi^{eff}) \\ \mathcal{G}: &\quad \psi^{eff}(x_1, \ldots, x_n) \\ &\quad D_{\mathcal{G}} = \{i_1, \ldots, i_m\} \end{aligned}$$

where $\psi^{eff}(x_1, \ldots, x_n)$ describes the desired goal models, and $D_{\mathcal{G}}$ represents the initial set of constants for the algorithm.

The composition approach is built on a notion of *beliefs*: a belief is captured as a set of *models* that are considered possible; i.e., at each point in a composition, our uncertainty about the true state of the execution is expressed in terms of the set of models that may be possible. An *initial belief* b_0 is created from the background ontology and the constants in $D_{\mathcal{G}}$. A *solved belief* is such a belief whose all models fulfill the desired goal effect.

Given a belief b and a service s, the result of *applying* s in b is a new belief (a set of models), denoted $apply(b, s)$. Each of the new models captures one possible way the old belief can be updated to reflect the service's effect ϕ^{eff}. Creating new models that satisfy a given logical expression is called *update reasoning*, and it is a known hard problem (cf. [47]). Hoffmann et al. use approximate reasoning with Horn theories, which they show to be tractable. The detailed formal definition of the function $apply(m, s)$ can be found in [44].

Fig. 8 outlines the composition algorithm, with updates from [44] to adopt WSMO-Lite terminology. The overall structure of the algorithm is typical for state-space search algorithms (see [48]). The inputs are the known services and the goal, and a successful output is a sequence of service applications that solves the goal. The algorithm searches in a space of beliefs that correspond to states in a typical AI planning search. The initial belief b_0, created on line 1, combines the background ontology with the goal data.

The algorithm works with a so-called *open-list* O, which contains all the beliefs that have yet to be processed; initially, that is only the belief s_0 (see line 3). In the open-list, each belief is kept in a 4-tuple $\langle b, h, H, p\rangle$, where b is the belief itself, p is the path that leads to this belief (a sequence of Web service applications), and h and H are additional values returned by a *heuristic function* that can help guide the search. The value of h is an estimate on how many Web services still need to be applied to b in order to obtain a solution (in other words, how close to the solution the belief appears to be), and H is a subset of all the available Web services that the heuristic function deems applicable to b. The heuristic value h guides the algorithm to the most appropriate beliefs in the search graph, and the set H prunes the search graph by dropping unwanted services.

The main loop on lines 4–11 executes until the open-list is empty, or until the solution is found. At every step, it selects the *best* next belief (as indicated by the heuristics) and removes it from the open-list (line 5). As presented, the algorithm is a "greedy best-first search" [48], but it can be changed effortlessly to other search algorithms, such as A* which is commonly very efficient in finding a solution.

Having selected the next belief, we compare it on line 6 with the goal using the function *isSolved*. If the goal effect is satisfied in the belief, we have found a solution and the algorithm ends.

If the solution has not been reached yet, we *expand* the currently-selected belief: we generate new beliefs by applying the available Web services (in the loop on lines 7–11). Applying a Web service s on a belief b (line 8) leads to a changed belief b', which is added to the open-list, along with its heuristic values and the updated path (lines 10 and 11). If the Web service s is not applicable to the belief b (either its precondition does not apply, or its effect leads to a contradiction), we simply skip this service and move to the next one (line 12).

If the open-list becomes empty, the algorithm has explored the entire search space without finding a solution, and it ends (line 12).

After the algorithm finds a composition solution, the composition can be presented to the user, who may need to fill in details related to data or process mediation (cf. [49,50]).

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

Algorithm: forward-search Web service composition for WSMO-Lite
Inputs: set S of known services annotated with capabilities,
$\psi^{eff}(x_1, \ldots, x_n)$ defining the goal models,
$D_{\mathcal{G}}$ with the initial constants.
Output: a list of Web services to be applied in a sequential composition

```
1  b_0 := initialBelief(D_G)
2  (h, H) := heuristicFunction(b_0)
3  O := (⟨b_0, h, H, ()⟩)
4  while O ≠ ∅
5  |  ⟨b, h, H, p⟩ := selectAndRemoveBest(O)
6  |  if isSolved(b) then return p
7  |  for each s ∈ H do
8  |  |  b' := apply(b, s)
9  |  |  if b' is undefined then next s
10 |  |  (h', H') := heuristicFunction(b')
11 |  |  add(O, ⟨b', h', H', add(p, s)⟩)
12 abort "no solution exists"
```

Fig. 8. Web service composition algorithm for WSMO-Lite.

7. Feasibility of the approach

Throughout this article, we have stressed the need for lightweight semantics for Web services, especially including RESTful APIs that have seen limited attention from Web service automation researchers. The languages we have presented directly address the need: WSMO-Lite covers service semantics, and hRESTS/MicroWSMO support RESTful APIs. In this section, we look at the feasibility of the approach from different angles: tooling support and viability (will it work?).

7.1. Tooling

WSMO-Lite is intended for automation of the use of Web services, thus the main implementation is in tools that realize automation algorithms such as those presented in Section 6. At OASIS, the main standardization body for WS–∗ Web services, such tools are called a Semantic Execution Environment (SEE),[28] which employs semantic technologies to support a user in achieving its goal with Web services.

A WSMO-Lite-based semantic execution environment, SOA4All Studio[29] was developed by the research project SOA4All.[30] It contains components for semantic service discovery, ranking, composition and monitoring.

Among other functionalities, SOA4All Studio supports service providers (or interested third parties) in creating semantic descriptions for Web services, both WS–∗ and RESTful. Editors for semantic service description are an important kind of implementation of WSMO-Lite. Describing Web services semantically is a knowledge-intensive task that cannot be fully automated without strong artificial intelligence, but tools such as SOA4All Studio's WSMO-Lite Editor/SOWER (for WS–∗ services)[31] and MicroWSMO Editor/SWEET (for RESTful services)[32] can ease the task by suggesting appropriate ontologies, guiding the user in applying semantic annotations to the underlying Web service descriptions:

- *WSMO-Lite Editor, a.k.a. SOWER*: supports adding (and manipulating) semantic annotations in WSDL and XML Schema descriptions, according to the distribution of the kinds of semantics defined in Table 3. Also supports the publication of new semantic descriptions in the SOA4All service registry, after transforming the WSDL into an RDF form of the WSMO-Lite minimal service model [51].
- *MicroWSMO Editor, a.k.a. SWEET*: supports adding and manipulating semantic annotations in the HTML documentation of RESTful services; therefore it also allows adding hRESTS annotations to which the semantic annotations can be attached. Also this editor enables the publication of the semantic descriptions in the registry, after transforming the hRESTS/MicroWSMO documents into the WSMO-Lite RDF form [52].

Editing tools may also verify some consistency and completeness criteria (we have proposed some such criteria for WSMO-Lite descriptions in [12]) and even guide a user through the steps of a semantic service description methodology. For instance, the OASIS SEE Technical Committee is working on *MEMOS* (A Methodology for Modeling Services [53]), which should be directly applicable to WSMO-Lite as well as other SWS frameworks.

SOA4All Studio is backed by iServe[33] [54], a semantic service description registry. iServe is fully based on the WSMO-Lite service model, but it is also capable of importing OWL-S service descriptions and treating them as WSMO-Lite descriptions. The registry publishes service descriptions in an open manner as linked data, and it provides a RESTful service discovery API that implements several discovery algorithms: (1) the functional classification discovery described in Section 6.1.1, (2) statistical text-similarity matchmaker based on iMatcher [55], and (3) a simple input/output subsumption matchmaker. Only the first algorithm is described in Section 6, as the adaptation of the other two to WSMO-Lite is straightforward.

7.2. Viability

The viability of the languages is demonstrated in a number of places throughout this article; let us summarize them here in the order of the life cycle of semantic service descriptions: descriptions are first created, then stored (published) somewhere, and finally processed for discovery and other automation purposes.

[28] http://oasis-open.org/committees/tc_home.php?wg_abbrev=semantic-ex.
[29] http://technologies.kmi.open.ac.uk/soa4all-studio/.
[30] http://soa4all.eu.
[31] http://technologies.kmi.open.ac.uk/soa4all-studio/provisioning-platform/sower/.
[32] http://technologies.kmi.open.ac.uk/soa4all-studio/provisioning-platform/sweet/.
[33] http://iserve.kmi.open.ac.uk/.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

Table 5
Summary of comparison of WSMO-Lite and WSMO.

	WSMO	OWL-S	WSMO-Lite
Scope			
Breadth	Broad (4 top-level areas)	Narrow (only services)	
Depth	Deeper	Shallower (no orchestration)	
Relation to standards			
Web services	Grounding in WSDL		SAWSDL
Semantic Web	Supplements RDF/OWL	Use RDF/OWL	
Syntax	"Human-readable", RDF, XML	RDF	
Other differences			
Svc. classification	Not supported	Deprecated	Supported
RESTful services	No support (but grounding possible)		Direct support
Tool support	Several existing tools		A few tools

While we have not performed a direct examination of the ease and usability of service description authoring, the creation and authoring of WSMO-Lite descriptions is likely to benefit from the principle of remaining lightweight that permeates our work. In Section 7.1, we have described two published editor tools that support SAWSDL and hRESTS/MicroWSMO. Authoring WSDL/SAWSDL descriptions is a well-known and settled area, but annotating the HTML documentation of RESTful services is still a research area with space for further exploration. For example, some RESTful APIs have documentation that is not a close fit to the hRESTS structure of a list of operations with their separate inputs and outputs, in which case an editor tool will require the use of the flexible RDFa form of hRESTS and MicroWSMO discussed in Section 5.3. The third-party development and use of the previously mentioned two editor tools, free of significant issues against our languages, has shown that the creation of WSMO-Lite descriptions can be effectively supported for both major kinds of Web services.

For the storage and publishing of WSMO-Lite service descriptions, Section 7.1 describes the public registry iServe. At the time of this writing, the registry contains almost 2000 service descriptions. iServe was used as the service registry in the EU projects SOA4All and NoTube [56].

Finally for processing of SWS descriptions, Section 6 describes several algorithms directly built on the WSMO-Lite service model and semantics. The discovery algorithms in particular are implemented in the iServe registry showing that they perform on par with state-of-the-art service matchmakers [57]. This gives us confidence that further SWS algorithms can be adapted to WSMO-Lite without loss of functionality, and that WSMO-Lite is therefore a viable SWS description approach. While adopting different existing SWS automation systems to WSMO-Lite does not immediately mean interoperability, it gives researchers a common ground and a shared vocabulary, making it easier to identify and work out the differences between various approaches to the same SWS automation problem (such as discovery).

8. Comparison with other SWS frameworks and related work

WSMO-Lite is a new development in the area of semantic descriptions of Web services. To our knowledge, WSMO-Lite is the first SWS description approach built on top of SAWSDL. The standardization of SAWSDL has spurred research in ways to connect earlier SWS frameworks with the new standard, see [58–60]. These efforts simply used SAWSDL model references to point to elements of OWL-S or WSMO service descriptions, in effect using SAWSDL for grounding. Even with SAWSDL groundings, WSMO and OWL-S semantic Web service descriptions stay conceptually independent of the underlying technical descriptions.

In contrast, WSMO-Lite builds semantic descriptions directly on the underlying technical descriptions (as per our design principles from Section 2). In Section 8.1, we compare WSMO-Lite to the two major preceding frameworks WSMO and OWL-S, showing that the most important differences are not in technical capabilities, but in positioning toward the Web service technologies on which service-oriented systems are built.

Additionally, in Section 8.2 we also compare WSMO-Lite to WSDL-S [5], the predecessor of SAWSDL. Since WSDL-S has several features that did not become parts of the SAWSDL standard, it is useful here to show how WSMO-Lite brings them back.

Finally, Section 8.3 briefly discusses other relevant related works, and in Section 8.4 we discuss the limitations of WSMO-Lite.

8.1. Comparing WSMO-Lite to WSMO and OWL-S

As summarized in Table 5, the main differences between WSMO-Lite and the two preceding frameworks lie especially in the scope of the frameworks and in their relation to standards, but there are other notable differences. All of the differences are detailed in the following paragraphs. Where OWL-S differs from WSMO, WSMO-Lite happens to be closer to OWL-S; therefore we generally start by comparing WSMO-Lite to WSMO and then add a remark about OWL-S.

The foremost difference between WSMO-Lite and WSMO is that of *scope*: where WSMO is a comprehensive framework that covers all the areas of semantic descriptions around services, OWL-S and WSMO-Lite have a narrower scope. In particular, WSMO contains language structures to define ontologies, user goals, and mediators, and OWL-S and WSMO-Lite do not.

Further, where WSMO details both the outer and the inner behavior of services in choreography and orchestration process descriptions, both OWL-S and WSMO-Lite only describe the outward behavior of services; OWL-S uses an explicit process model,[34] while WSMO-Lite captures the functionalities of the operations, leaving the choreography process implicit. In this regard, the scope of WSMO is deeper.

A major difference between WSMO, OWL-S and WSMO-Lite, one which may affect the adoption of these semantic technologies in service-oriented environments, lies in the *relation* of the technologies *to existing standards*.

From the viewpoint of standards for Web services, WSMO-Lite builds directly on WSDL and SAWSDL, while both WSMO and OWL-S remain independent of Web service technologies, shielded by a layer called "grounding" that provides the necessary links to WSDL.

In relation to the Semantic Web standards RDF and OWL, both WSMO-Lite and OWL-S use them directly, while WSMO provides its own ontology language, with a mapping to the W3C Recommendations.

In both areas of standardization, WSMO-Lite is positioned close to the formal standards; as such, it can easily be adopted in environments that already use the standard technologies.

The WSMO framework comes with a special *syntax*, described as "abstract" and "human-readable", and it also provides two further exchange formats, in XML and in RDF. The human-readable syntax is especially intended for advanced users who can author logical statements and service descriptions in a source form; other users are expected to use authoring tools. In contrast, both OWL-S and WSMO-Lite use RDF as their only representation format.

Other notable differences between WSMO-Lite and the two preceding frameworks lie in support for using functionality classifications, support for RESTful services, and implementations within tools.

[34] OWL-S Process Model does not make an explicit conceptual differentiation between choreography and orchestration, and primarily focuses on the outward behavior of services. While it does exhibit internal features through procedural control constructs, it is relatively modest in terms of language constructs compared to other fully fledged process orchestration languages.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

As part of the functional and behavioral semantics, WSMO-Lite supports the use of *functionality-based classifications*. WSMO focuses on expressing service functionality and behavior through logical expressions, and therefore it has no explicit support for categorizing service and operation functionalities. In OWL-S, service classification used to be supported (only on services, not on operations); in the newest version this support is deprecated and delegated to domain-specific extensions.

In relation to the hRESTS/MicroWSMO microformats, WSMO-Lite *supports* the semantic description of *RESTful services*, which are an increasingly important part of the Web. In contrast, both OWL-S and WSMO would need a grounding specification for RESTful services, and we know of no efforts in this direction.

Finally, while there are already existing tools for WSMO (e.g. WSMX, WSMT, and IRS-III[35]) and for OWL-S (e.g. OWL-S Editor, OWL-S Matcher, and Web Service Composer[36]), comparatively fewer tools have been developed that support WSMO-Lite, as we have discussed in Section 7.1.

In summary, WSMO-Lite is positioned as an extension of the sole SWS standard SAWSDL, with a tight scope and a close relation to underlying technologies. WSMO-Lite's direct support for RESTful services is a major advantage over preceding SWS approaches. On the other hand, there are only a few tools that support WSMO-Lite service description authoring and processing.

8.2. Comparing WSMO-Lite to WSDL-S

WSDL-S is a technology that was developed as an extension of WSDL, in order to bring semantics closer to the underlying Web services technologies. Along with a generic *model reference* construct that became the cornerstone of SAWSDL, WSDL-S also defined constructs for WSDL interface categorization and for operation preconditions and effects, which were not carried over into SAWSDL.

To facilitate semantic automation, WSDL-S allows the expression of preconditions and effects on WSDL operations. In WSMO-Lite, preconditions and effects are treated as functional annotations, supported both on operations, as part of the behavioral semantics of a service, and on the service itself, as part of its functional semantics. Specifying high-level preconditions and effects on the service as a whole is useful for coarse-grained service discovery and composition, without delving into the details of the service's operations. In this way, the discovery and composition process may be able to overlook formal incompatibilities that SWS automation is not able to resolve, but which can be overcome through process mediation devised by a human engineer.

For coarse-grained service discovery, WSDL-S also specifies a construct for attaching categorization information to WSDL interfaces. WSMO-Lite treats categorizations as functionality classifications, supported both on the service (interface) as functional semantics, and on the service's operations, where functional categories serve as parts of the behavioral semantics of the service. Where WSDL-S expects the coarse-grained approach of categorizations to be useful only on the level of whole services, WSMO-Lite can also handle categories applicable to operations; for instance wsdlx:SafeInteraction is a category of operations that are *safe* for invocation, as defined in the architecture of the Web, and discussed here in Section 5.2.

Effectively, WSMO-Lite embraces all WSDL-S constructs, generalizing their use as part of the functional and behavioral semantics of Web services.

8.3. Other relevant related works

The Unified Service Description Language (USDL) [61] is a general purpose domain-independent service description language. It aims to capture a comprehensive description of services, across various facets, business aspects, and perspectives of the wide variety of stakeholders involved in a service ecosystem. As such, it is independent of any technologies such as WS–∗ or REST. [62] proposed a mechanism to annotate WSDL services with USDL descriptions. More recently, [63] proposed a comprehensive vocabulary for capturing and sharing USDL service based on Linked Data principles. Despite these attempts, to our knowledge, USDL has not been used to comprehensively reconcile the WS–∗ and RESTful worlds.

Semantic descriptions of REST APIs is an active research field. A recent survey of existing approaches is provided in [64] where approaches such as WADL, RDFa, ROSM, SPARQL-Based descriptions, SEREDASj, etc., are discussed. None of these approaches is primarily focused on reconciling WS–∗ and RESTful services.

8.4. Limitations of WSMO-Lite

WSMO-Lite is very lightweight and extensible, therefore it is not easy to pinpoint its limitations. Nevertheless, here we attempt to identify limitations that could be encountered in future work on top of WSMO-Lite, even if they have not yet been experienced in implementations:

- Message is an atomic construct. If we consider an event scheduling operation whose output is an Event, it may contain information about its attendees that might be annotated as Person. WSMO-Lite provides no mechanism to capture that the Person is a part of the Event in this case. We believe this type of data modeling should better be addressed in the ontology used for annotations (where an Event can already point to its attendees), so the WSMO-Lite message could only be annotated with the Event class.
- Behavioral semantics are represented as functional annotations of operations. There may be cases where the annotations should be accompanied by additional context information, such as for example an explicit declaration of shared variables.
- We put nonfunctional semantics on the service as a whole; if need be, WSMO-Lite can be extended straightforwardly to support nonfunctional semantics on individual operations.
- MicroWSMO may encounter modeling difficulties if the HTML structure of the documentation of some API does not permit a proper nesting of the microformat-annotated HTML elements. This is a limitation shared with microformats in general; the problem can be overcome by refactoring the HTML code, which can usually be achieved without affecting the presentation of the documentation.
- At the moment, WSMO-Lite has no support for other service descriptions such as WADL or the JSON-based Swagger.[37] This present limitation would constitute future work, prioritized by the popularity the languages.

9. Conclusions and outlook

The Semantic Web of data is starting to be taken seriously outside the research community, as demonstrated by the increasing numbers of data providers in the Linked Data project [65] and take

35 http://wsmx.org/, http://wsmt.sf.net/, http://technologies.kmi.open.ac.uk/irs/.

36 http://semanticweb.org/wiki/OWL-S_Editor, http://owlsm.projects.semwebcentral.org/, http://www.mindswap.org/~evren/composer/.

37 http://swagger.wordnik.com.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006

up by major players such as Google, Facebook and Microsoft. However, the Semantic Web vision is not limited to data interoperability; it has also always included processing services, as implied in [66] and further discussed in [67].

Research on Semantic Web Services strives for automating the tasks associated with using and combining Web services. Several approaches to semantic Web service automation have been proposed. However, SWS research has been fragmented and detached from the lower-level WS–∗ specifications, which can explain its limited adoption in industrial settings. W3C has begun to consolidate SWS approaches through standardization, with SAWSDL being the first step, albeit a small one. SAWSDL directly addresses the issue of limited adoption by putting semantics close to the accepted standard Web services description language WSDL. But SAWSDL does not specify any actual service semantics.

In this article, we have introduced *WSMO-Lite*, an ontology for service semantics that fits directly into SAWSDL annotations, and to demonstrate the viability of this ontology, we have adapted *several SWS automation algorithms* to WSMO-Lite. WSMO-Lite is intentionally lightweight, in order to ease the learning curve for adopters of SWS technologies.

In addition to SAWSDL-based support of WS–∗ services, WSMO-Lite also supports *RESTful services*, which have so far received relatively little attention from the SWS research community[38] (compared to the larger amount of research work that went into SWS for WS–∗ services). RESTful services are an increasingly important component of Web applications. There is no widely accepted machine-oriented description language for RESTful services, therefore we also propose two microformats, *hRESTS* and *MicroWSMO*, which mirror WSDL and SAWSDL which augment human-oriented HTML documentation of RESTful services. With a minimal semantic service model that is an abstraction of WSDL and hRESTS, RESTful services can be included in semantic processing with WSMO-Lite seamlessly. The easy integration of RESTful and WS–∗ Web services will especially gain importance as the popularity of RESTful services increases in enterprise environments that have traditionally favored WS–∗ technologies.

To summarize the intended impact of WSMO-Lite:

- WSMO-Lite brings a unifying approach for semantics addressing both WS–∗ and RESTful services. While lightweight, it supports well-known automation algorithms.
- WSMO-Lite aims to simplify the creation of semantic descriptions of Web services, and thus to improve the usage of Web services.

The main task for future work is fostering the adoption of SAWSDL and WSMO-Lite in industrial WS–∗ settings, and the adoption of hRESTS and MicroWSMO in RESTful service-oriented systems. Adoption is related to standardization, especially in the standards-heavy environment of service-oriented computing. The standardization of SWS approaches can continue in a piece-meal fashion as demonstrated by SAWSDL; WSMO-Lite has already been submitted to W3C as a Member Submission [73]. The W3C Team Comment on the submission stated that it "is a useful addition to SAWSDL for annotations of existing services and the combination of both techniques can certainly be applied to a large number of semantic Web services use cases".

Furthermore, adapting various discovery and composition algorithms to WSMO-Lite and performing performance evaluations of their implementations adapted to WSMO-Lite is also part of future work. For example, discovery evaluation can be performed within the framework of the S3 Contest on Semantic Service Selection [74][39] which is the reference contest for evaluating service matchmakers.[40]

Acknowledgments

The authors would like to thank Dr. Karthik Gomadam of the Kno.e.sis Center for his contributions to the microformat hRESTS; James Scicluna of STI Innsbruck for his help in adapting their Web service composition and ranking approaches to WSMO-Lite; and Dr. Dong Liu of the Knowledge Media Institute (The Open University) for his help with testing our semantic models within iServe.

References

[1] K. Sycara, M. Paolucci, A. Ankolekar, N. Srinivasan, Automated discovery, interaction and composition of semantic Web services, Web Semant. Sci. Serv. Agents World Wide Web 1 (1) (2003) 27–46.
[2] OWL-S 1.1 Release, Tech. Rep., OWL Services Coalition, November 2004. Available at: http://www.daml.org/services/owl-s/1.1/.
[3] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, D. Fensel, Web service modeling ontology, Appl. Ontol. 1 (1) (2005) 77–106.
[4] Web Services Description Language (WSDL) Version 2.0, Recommendation, W3C, June 2007. Available at: http://www.w3.org/TR/wsdl20/.
[5] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma, Web Service Semantics—WSDL-S, Technical Note, April 2005. Available at: http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html.
[6] SOAP Version 1.2 Part 1: Messaging Framework, Recommendation, W3C, June 2003. Available at: http://www.w3.org/TR/soap12-part1/.
[7] Semantic Annotations for WSDL and XML Schema, Recommendation, W3C, August 2007. Available at: http://www.w3.org/TR/sawsdl/.
[8] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly Media, 2007.
[9] Hypertext Transfer Protocol—HTTP/1.1, RFC 2616, IETF, June 1999. Available at: http://www.rfc-editor.org/rfc/rfc2616.txt.
[10] Architecture of the World Wide Web, Recommendation, W3C, December 2004. Available at: http://www.w3.org/TR/webarch/.
[11] C. Pautasso, O. Zimmermann, F. Leymann, RESTful Web services vs. big Web services: making the right architectural decision, in: Proceedings of the 17th International World Wide Web Conference, WWW2008, 2008.
[12] T. Vitvar, J. Kopecký, J. Viskova, D. Fensel, WSMO-lite annotations for Web services, in: The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Springer, Tenerife, Spain, 2008.
[13] J. Kopecký, K. Gomadam, T. Vitvar, hRESTS: an HTML microformat for describing RESTful Web services, in: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence, WI-08, Sydney, Australia, 2008.
[14] R. Khare, T. Çelik, Microformats: a pragmatic path to the semantic Web (Poster), in: Proceedings of the 15th International Conference on World Wide Web, 2006, pp. 865–866.
[15] M.J. Hadley, Web application description language (WADL), Tech. Rep., Sun Microsystems, 2006. Available at: https://wadl.dev.java.net/.
[16] RDF Vocabulary Description Language 1.0: RDF Schema, Recommendation, W3C, February 2004. Available at: http://www.w3.org/TR/rdf-schema/.
[17] H. Chesbrough, J. Spohrer, A research manifesto for services science, Commun. ACM 49 (7) (2006) 35–40.
[18] Z. Baida, J. Gordijn, B. Omelayenko, A shared service terminology for online service provisioning, in: ICEC'04: Proceedings of the 6th International Conference on Electronic Commerce, ACM Press, New York, NY, USA, 2004.
[19] C. Preist, A conceptual architecture for semantic Web services, in: Proceedings of the 3rd International Semantic Web Conference, ISWC 2004, in: Lecture Notes in Computer Science, LNCS, vol. 3298, Springer, 2004.
[20] R. Ferrario, N. Guarino, Towards an ontological foundation for services science, in: Proceedings of FIS 2008, 2008.
[21] Reference Model for Service Oriented Architecture 1.0, OASIS Standard, October 2006. Available at: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html.
[22] Web Services Architecture, Working Group Note, W3C, February 2004. Available at: http://www.w3.org/TR/ws-arch.
[23] A.P. Sheth, Semantic Web process lifecycle: role of semantics in annotation, discovery, composition and orchestration, in: Invited Talk, Workshop on E-Services and the Semantic Web, 2003. Available at: http://lsdis.cs.uga.edu/lib/presentations/WWW2003-ESSW-invitedTalk-Sheth.pdf.
[24] Web Services Description Language (WSDL) Version 2.0: Adjuncts, Recommendation, W3C, June 2007. Available at: http://www.w3.org/TR/wsdl20-adjuncts/.

38 Though relatively few, approaches such as [68–72] for matchmaking of RESTful service have recently started to appear.

39 http://www-ags.dfki.uni-sb.de/klusch/s3/.

40 See also [75] for a recent survey on service discovery.

[25] Web Services Description Language (WSDL) Version 2.0: RDF Mapping, Working Group Note, W3C, June 2007. Available at: http://www.w3.org/TR/wsdl20-rdf.
[26] R.T. Fielding, Architectural styles and the design of network-based software architectures (Ph.D. thesis), Richard N. Taylor, University of California, Irvine, Chair, 2000.
[27] OWL Web Ontology Language Overview, Recommendation 10 February 2004, W3C, 2004. Available at: http://www.w3.org/TR/owl-features/.
[28] RIF Core Dialect, Recommendation, W3C, June 2010. Available at: http://www.w3.org/TR/rif-core/.
[29] Simple Knowledge Organization System, Recommendation, W3C, August 2009. Available at: http://www.w3.org/TR/skos-reference/.
[30] J. Kopecký, T. Vitvar, D. Fensel, WSMO-Lite: lightweight semantic descriptions for services on the Web, CMS WG Working Draft D11, March 2009. Available at: http://cms-wg.sti2.org/TR/d11/.
[31] M. Maleshkova, C. Pedrinaci, J. Domingue, Investigating Web APIs on the World Wide Web, in: Proceedings of he 8th IEEE European Conference on Web Services, ECOWS 2010, 2010. Available at: http://oro.open.ac.uk/24320/.
[32] A. Polleres, T. Krennwallner, N. Lopes, J. Kopecký, S. Decker, XSPARQL Language Specification, W3C member submission, January 2009. Available at: http://www.w3.org/Submission/xsparql-language-specification/.
[33] A.P. Sheth, K. Gomadam, J. Lathem, SA-REST: semantically interoperable and easier-to-use services and mashups, IEEE Internet Comput. 11 (6) (2007) 91–94.
[34] RDFa in XHTML: Syntax and Processing, Recommendation, W3C, October 2008. Available at: http://www.w3.org/TR/rdfa-syntax/.
[35] Gleaning Resource Descriptions from Dialects of Languages (GRDDL), Recommendation, W3C, September 2007. Available at: http://www.w3.org/TR/grddl/.
[36] HTML 4.01 Specification, Recommendation, W3C, December 1999. Available at: http://www.w3.org/TR/html401.
[37] J. Kopecký, E. Simperl, Semantic Web service offer discovery for E-commerce, in: Proceedings of the 10th International Conference on Electronic Commerce 2008, Innsbruck, Austria, August 19–22, 2008.
[38] M. Klusch, Semantic Web service coordination, in: M. Schumacher, H. Helin, H. Schuldt (Eds.), CASCOM: Intelligent Service Coordination in the Semantic Web, Springer Birkhäuser, 2008 (Chapter 4).
[39] U. Keller, R. Lara, H. Lausen, D. Fensel, Semantic Web service discovery in the WSMO framework, in: J. Cardoso (Ed.), Semantic Web: Theory, Tools and Applications, Idea Publishing Group, 2006.
[40] U. Keller, R. Lara, A. Polleres, I. Toma, M. Kifer, D. Fensel, D5.1: WSMO Web Service Discovery, Tech. Rep., DERI Innsbruck, 2004. Available from http://www.wsmo.org/TR/d5/d5.1.
[41] M. Stollberg, Scalable semantic Web service discovery for goal-driven service-oriented architectures (Ph.D. thesis), University of Innsbruck, 2008.
[42] S. Dustdar, W. Schreiner, A survey on Web services composition, Int. J. Web Grid Serv. 1 (1) (2005) 1–30.
[43] F. Lecue, A. Leger, A formal model for semantic Web service composition, Lecture Notes in Comput. Sci. 4273 (2006) 385–398.
[44] J. Hoffmann, I. Weber, J. Scicluna, T. Kaczmarek, A. Ankolekar, Combining scalability and expressivity in the automatic composition of semantic Web services, in: Proceedings of the 8th International Conference on Web Engineering, ICWE'08, Yorktown Heights, USA, 2008.
[45] M. Pistore, P. Roberti, P. Traverso, Process-level composition of executable Web services: on-the-fly versus "Once-for-all" composition, in: Proceedings of the Second European Semantic Web Conference, ESWC'05, in: Lecture Notes in Computer Science, LNCS, vol. 3532, Springer-Verlag, 2005.
[46] M. Winslett, Reasoning about actions using a possible models approach, in: Proc. AAAI'88, 1988.
[47] A. Herzig, O. Rifi, Propositional belief base update and minimal change, Artificial Intelligence 115 (1) (1999) 107–138.
[48] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, second ed., Prentice Hall, Englewood Cliffs, NJ, 2002.
[49] E. Cimpian, A. Mocan, M. Stollberg, Mediation enabled semantic Web services usage, in: The Semantic Web—ASWC 2006, in: Lecture Notes in Computer Science, LNCS, vol. 4185, Springer, 2006.
[50] A. Mocan, E. Cimpian, An ontology-based data mediation framework for semantic environments, Int. J. Sem. Web Inf. Syst. 3 (2) (2007) 69–98.
[51] M. Maleshkova, G.A. Rey, A. Simov, B. Renie, D. Liu, Service provisioning platform second prototype, Deliverable D2.1.4 of the Project SOA4All, August 2010. Available at: http://soa4all.eu/file-upload.html?func=startdown&id=229.
[52] M. Maleshkova, C. Pedrinaci, J. Domingue, Semantic annotation of Web APIs with SWEET, in: Proceedings of the 6th Workshop on Scripting and Development for the Semantic Web, Colocated with the Extended Semantic Web Conference, Heraklion, Greece, 2010.
[53] M. Kerrigan, B. Norton, E. Simperl, MEMOS: a methodology for modeling services, in: Proceedings of the 5th International Workshop on Semantic Business Process Management, SBPM, Crete, Greece, 2010, Co-located with the 7th European Semantic Web Conference, ESWC.
[54] C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecký, J. Domingue, iServe: a linked services publishing platform, in: Proceedings of 1st International Workshop on Ontology Repositories and Editors for the Semantic Web, ORES 2010, Colocated with 7th ESWC, 2010.
[55] C. Kiefer, A. Bernstein, The creation and evaluation of iSPARQL strategies for matchmaking, in: Proceedings of the 5th European Semantic Web Conference (ESWC), in: Lecture Notes in Computer Science, LNCS, vol. 5021, Springer, 2008.
[56] H.Q. Yu, N. Benn, S. Dietze, R. Siebes, C. Pedrinaci, D. Liu, D. Lambert, J. Domingue, Two-staged approach for semantically annotating and brokering TV-related services, in: Proceedings of the IEEE International Conference on Web Services, ICWS, Miami, Florida, USA, 2010.
[57] J. Kopeckỳ, Web service automation supported by lightweight semantic annotations (Ph.D. thesis), University of Innsbruck, 2012.
[58] D. Martin, M. Paolucci, M. Wagner, Bringing semantic annotations to Web services: OWL-S from the SAWSDL perspective, in: K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (Eds.), The Semantic Web, in: Lecture Notes in Computer Science, LNCS, vol. 4825, Springer, 2007.
[59] M. Paolucci, M. Wagner, D. Martin, Grounding OWL-S in SAWSDL, in: B.J. Krämer, K.-J. Lin, P. Narasimhan (Eds.), ICSOC, in: Lecture Notes in Computer Science, LNCS, vol. 4749, Springer, 2007.
[60] J. Kopecký, M. Moran, T. Vitvar, D. Roman, A. Mocan, WSMO Grounding, April 2007. Available at: http://www.wsmo.org/TR/d24/d24.2/.
[61] J. Cardoso, A. Barros, N. May, U. Kylau, Towards a unified service description language for the Internet of services: requirements and first developments, in: 2010 IEEE International Conference on Services Computing (SCC), IEEE, 2010.
[62] S. Kona, A. Bansal, L. Simon, A. Mallya, G. Gupta, USDL: a service-semantics description language for automatic service discovery and composition, Int. J. Web Serv. Res. (IJWSR) 6 (1) (2009) 20–48.
[63] C. Pedrinaci, J. Cardoso, T. Leidig, Linked USDL: a vocabulary for Web-scale service trading, in: The Semantic Web: Trends and Challenges, Springer, 2014, pp. 68–82.
[64] R. Verborgh, A. Harth, M. Maleshkova, S. Stadtmüller, T. Steiner, M. Taheriyan, R. Van de Walle, Survey of semantic description of rest APIs, in: C. Pautasso, E. Wilde, R. Alarcon (Eds.), REST: Advanced Research Topics and Practical Applications, Springer, New York, 2014, pp. 69–89. URL: http://dx.doi.org/10.1007/978-1-4614-9299-3_5.
[65] C. Bizer, T. Heath, T. Berners-Lee, Linked data—the story so far, Int. J. Sem. Web Inf. Syst. (IJSWIS) 5 (3) (2009) 1–22.
[66] T. Berners-Lee, J. Hendler, O. Lassila, The semantic Web, Sci. Am. 284 (5) (2001) 34–43.
[67] J. Hendler, T. Berners-Lee, E. Miller, Integrating applications on the semantic Web, J. Inst. Electr. Eng. Japan 122 (10) (2002) 676–680 (in Japanese); English reprint. Available at: http://www.w3.org/2002/07/swint.
[68] F. Slaimi, S. Sellami, O. Boucelma, A. Ben Hassine, Flexible matchmaking for RESTful Web services, in: R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. De Leenheer, D. Dou (Eds.), On the Move to Meaningful Internet Systems: OTM 2013 Conferences, in: Lecture Notes in Computer Science, vol. 8185, Springer, Berlin, Heidelberg, 2013, pp. 542–554. URL: http://dx.doi.org/10.1007/978-3-642-41030-7_39.
[69] M. Pantazoglou, A. Tsalgatidou, A generic query model for the unified discovery of heterogeneous services, IEEE Trans. Serv. Comput. 6 (2) (2013) 201–213.
[70] U. Lampe, S. Schulte, M. Siebenhaar, D. Schuller, R. Steinmetz, Adaptive matchmaking for RESTful services based on hRESTS and microWSMO, in: Proceedings of the 5th International Workshop on Enhanced Web Service Technologies, WEWST'10, ACM, New York, NY, USA, 2010, URL: http://doi.acm.org/10.1145/1883133.1883136.
[71] L. Panziera, M. Comerio, M. Palmonari, F. De Paoli, Distributed matchmaking and ranking of Web APIs exploiting descriptions from Web sources, in: Service-Oriented Computing and Applications, SOCA, 2011 IEEE International Conference on, 2011.
[72] L. Panziera, M. Comerio, M. Palmonari, F. De Paoli, C. Batini, Quality-driven extraction, fusion and matchmaking of semantic Web API descriptions, J. Web Eng. 11 (3) (2012) 247.
[73] D. Fensel, F. Fischer, J. Kopecký, R. Krummenacher, D. Lambert, T. Vitvar, WSMO-lite: lightweight semantic descriptions for services on the Web, W3C member submission, August 2010. Available at: http://www.w3.org/Submission/WSMO-Lite/.
[74] M. Klusch, Overview of the S3 contest: performance evaluation of semantic service matchmakers, in: Semantic Web Services, Springer, 2012, pp. 17–34.
[75] M. Klusch, Service discovery, in: R. Alhajj, J. Rokne (Eds.), Encyclopedia of Social Networks and Mining (ESNAM), Springer, 2014.

Please cite this article in press as: D. Roman, et al., WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs, Web Semantics: Science, Services and Agents on the World Wide Web (2014), http://dx.doi.org/10.1016/j.websem.2014.11.006